Naysayer Proofs

István András Seres^{*†1}, Noemi Glaeser^{*‡2,3}, and Joseph Bonneau^{§4,5}

¹Eötvös Loránd University ²University of Maryland ³Max Planck Institute for Security and Privacy ⁴a16z crypto research ⁵New York University

January 28, 2025

Abstract

This work introduces the notion of naysayer proofs. We observe that in numerous (zero-knowledge) proof systems, it is significantly more efficient for the verifier to be convinced by a so-called naysayer that a false proof is invalid than it is to check that a genuine proof is valid. We show that every NP language has logarithmic-size and constant-time naysayer proofs. We also show practical constructions for several example proof systems, including FRI polynomial commitments, post-quantum secure digital signatures, and verifiable shuffles. Naysayer proofs enable an interesting new optimistic verification mode potentially suitable for resource-constrained verifiers, such as smart contracts.

1 Introduction

In most blockchains with programming capabilities, e.g., Ethereum $[W^+14]$, developers are incentivized to minimize the storage and computation complexity of on-chain programs. Applications with high compute or storage incur significant fees, commonly referred to as *gas*, to compensate validators in the network. Often, these costs are passed on to users of an application.

High gas costs have motivated many applications to utilize *verifiable computation* [GGP10], off-loading expensive operations to powerful but untrusted off-chain entities who perform arbitrary computation and provide a succinct non-interactive proof (SNARK) that the claimed result is correct. This computation can even depend on secret inputs not known to the verifier in the case of zero-knowledge proofs (i.e., zkSNARKs).

Verifiable computation leads to a paradigm in which smart contracts, while capable of arbitrary computation, primarily act as verifiers and outsource all significant computation off-chain. A motivating application is *rollups*, which combines transactions from many users into a single smart contract which verifies a proof that all have been executed correctly. However, verifying these proofs can still be costly. For example, the StarkEx rollup has spent hundreds of thousands of dollars to date to verify FRI polynomial commitment opening proofs.¹

We observe that this proof verification is often wasteful. In most applications, provers have strong incentives to only post correct proofs, suffering direct financial penalties (in the form of a lost security deposit) or indirect costs to their reputation and business for posting incorrect proofs. As a result, a significant fraction of a typical layer-1 blockchain's storage and computation is expended verifying proofs, which are almost always correct.²

This state of affairs motivates us to propose a new paradigm called *naysayer proofs*. In this paradigm, the verifier (e.g., a rollup smart contract) optimistically accepts a submitted proof without verifying its correctness. Instead, any observer can check the proof off-chain and, if needed, prove its *incorrectness* to the verifier by submitting a *naysayer proof*. The verifier then checks the naysayer proof and, if it is correct, rejects the original proof. Otherwise, if no

^{*}István András Seres and Noemi Glaeser contributed equally to this work.

 $^{^{\}dagger}$ seresistvanandras@gmail.com. The majority of this work was done at a16z crypto research.

[‡]nglaeser@umd.edu. The majority of this work was done at a16z crypto research.

[§]jbonneau@gmail.com

¹https://etherscan.io/address/0x3e6118da317f7a433031f03bb71ab870d87dd2dd

 $^{^{2}}$ At the time of this writing, we are unaware of any major rollup service which has posted an incorrect proof in production.

	VC	fraud proof (interactive)	fraud proof (non-interactive)	naysayer proof
No optimistic assumption	•	0	0	0
Non-interactive	•	\bigcirc	lacksquare	igodot
Off-chain f	۲	•	lacksquare	•
Off-chain П.Verify	\bigcirc	-	-	•
Witness-independent challenge	-	0	0	•
Witness-independent resolution	•	igodot	0	•
No П.Prove	\bigcirc	•	•	0

Table 1: Trade-offs between VC, fraud proofs, and naysayer proofs.

party can successfully naysay the original proof before the end of the dispute period, the original proof is accepted. To deter denial of service, naysayers may be required to post collateral, which is forfeited if their naysayer proof is incorrect.

This paradigm potentially saves the verifier work in two ways. First, in the optimistic case, where the proof is not challenged, the verifier does no work at all. We expect this to almost always be the case in practice. Second, even in the pessimistic case, checking the naysayer proof may be much more efficient than checking the original proof (e.g., if the verification fails at a single point, the naysayer proof can just point to that specific step). In other words, the naysayer acts as a helper to the verifier by reducing the cost of the verification procedure in fraudulent cases. At worst, checking the naysayer proof is equivalent to verifying the original proof (this is the trivial naysayer construction).

Naysayer proofs enable other interesting trade-offs. For instance, naysayer proofs might be run at a lower security level than the original proof system. A violation of the naysayer proof system's soundness undermines the *completeness* of the original proof system. For an application like a rollup service, this results only in a loss of liveness; importantly, the rollup users' funds would remain secure. Liveness could be restored by falling back to full proof verification.

In Section 3, we formally define naysayer proofs and show that every NP language has a logarithmic size and constant-time naysayer proof. Before that, we discuss related work in Section 2. In Section 4, we provide concrete examples where naysayer proofs offer significant speedups. We discuss storage considerations in Section 5 and conclude with open research questions in Section 6.

2 Related Work

A concept related to the naysayer paradigm is *referred delegation* [FK97]. The idea has found widespread adoption [TR19, KGC⁺18, AAB⁺24] under the name "fraud proofs" or "fault proofs" and is the core idea behind *optimistic rollups* [Eth23b, Lab23, Opt23]. Like naysayer proofs, fraud proofs work under an optimistic assumption, i.e., a computation is assumed to be correct unless some party challenges it. In the context of optimistic rollups, a "prover" performs the computation off-chain and posts the result on-chain, where it is provisionally accepted. Any party can then challenge the correctness of the result by posting a challenge on-chain and engaging in a *bisection protocol* with the prover via on-chain messages. Once the problematic step is identified, it is re-executed on-chain to resolve the dispute. A dispute can also be resolved *non-interactively* by re-running the entire computation on-chain in the event of a dispute, an approach initially taken by Optimism [Sin22, Buc21]. If no one challenges the prover's result before the end of the *challenge period* (typically 7 days [Fic]), it is accepted and irreversibly committed on the layer-1 chain.

The naysayer approach offers significant speedups for the challenger over fraud proofs, since for succinct proof systems, verification is much more efficient than the original computation. Notice that there is a slight semantic difference between fraud proofs and naysayer proofs: A fraud proof challenges the correctness of the prover's *computation*, and thus can definitively show that the computation output is incorrect. In contrast, a naysayer proof challenges the correctness of the *accompanying proof*, and can therefore only show that the proof is invalid—the computation itself may still have been correct. A prover who performs the computation honestly has no incentive to attach an incorrect proof³, since that would mean it wasted computational power to compute the result, but would forfeit the reward (and likely incur some additional penalty).

³It is possible that an honest prover will still attach an incorrect proof if, for example, the proof generation software has a bug.

We compare classic verifiable computation, fraud proofs, and our new approach in Table 1. We discuss the main differences in more detail below.

- Assumptions. Both fraud proofs and naysayer proofs work under an optimistic assumption, meaning a computation is accepted as correct unless some party challenges it. This requires assuming that at least one honest party is present to challenge any incorrect results. This party must also be *able* to submit a challenge, meaning we rely on the censorship-resistance of the underlying blockchain and assume new messages are added within some known, bounded delay. VC does not rely on these assumptions since every computation is checked at the time the result is submitted. It is well known that this leads to the faster on-chain finality of zk-rollups, which use the VC paradigm and thus do not require a challenge period.
- **On-chain interaction.** Except for the interactive version of fraud proofs, all of the approaches require only a single message from the (off-chain) prover or challenger to the (on-chain) verifier. VC offers the strongest form of non-interactivity, since it consists of one round *total* (for the original computation and (non-existent) challenge phase). At the other end of the spectrum, optimistic rollups almost universally employ interactive fraud proofs, requiring multiple on-chain messages in case of a dispute. This means the challenge period must be long enough to ensure that all the messages of the dispute resolution protocol can be posted on-chain, even in the presence of some fraction of malicious consensus nodes who delay inclusion of the challenger's (or prover's) messages. We conjecture that by virtue of having a non-interactive challenge phase, naysayer proofs (and non-interactive fraud proofs) admit a shorter challenge period. Furthermore, the challenge period must also be long enough to accommodate the challenge resolution protocol to run on-chain. Thus, naysayer proofs should have an advantage even over non-interactive fraud proofs, since for all practical use cases, the on-chain resolution of the former (verifying a naysayer proof) will always be faster than re-computing the function *f* on-chain.
- **On-chain computation & witnesses.** As is their goal, none of the approaches require running the original computation f on-chain, except for non-interactive fraud proofs in the (rare) case of a dispute. Compared to VC, fraud proofs and naysayer proofs do not require running proof verification on-chain (fraud proofs do not use a proof system at all). However, fraud proofs require the full computation input (including any off-chain input w, which we refer to as the witness) to be available to potential challengers and at least in part to the verifier. Neither VC nor naysayer proofs require this information to verify the correctness of the output y': they use only the statement and proof, which are already available on-chain.
- **Underlying proof system.** Finally, a major advantage of fraud proofs is that they do not use any proof system at all. This makes them much easier to implement and deploy. VC and naysayer proofs, on the other hand, require computing a succinct proof, which is costly both in terms of implementation complexity and prover runtime. However, the design and efficiency of the bisection protocol can depend significantly on the programming model used [KGC⁺18] and the particular function f being computed [PD16, PB17, SNBB19, SJSW19]. We thus view naysayer proofs as a drop-in replacement for the many application-specific fault proofs, offering an alternative which is both more general and more efficient.

3 Naysayer Proofs

In this section, we introduce our system model and the syntax of naysayer proofs and show that logarithmic-size and constant-time verifiable (i.e., succinct) naysayer proofs exist for all proof systems with polynomial-time verification. First, we recall the syntax of non-interactive (zero-knowledge) proofs. We refer the reader to [Tha23] for a formal description of the properties of NIZKs (e.g., correctness, soundness, zero-knowledge).

Definition 1 (Non-interactive proof). A non-interactive proof Π for some NP language \mathcal{L} (with the corresponding relation $\mathcal{R}_{\mathcal{L}}$) is a tuple of PPT algorithms (Setup, Prove, Verify):

- $\mathsf{Setup}(1^{\lambda}) \to \mathsf{crs:}\ Given\ a\ security\ parameter,\ output\ a\ common\ reference\ string\ \mathsf{crs.}\ This\ algorithm\ might\ use\ private\ randomness\ (a\ trusted\ setup).$
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi: \text{ Given the crs, an instance } x, \text{ and witness } w \text{ such that } (x, w) \in \mathcal{R}_{\mathcal{L}}, \text{ output a proof } \pi.$
- Verify(crs, x, π) \rightarrow {0,1}: Given the crs and a proof π for the instance x, output a bit indicating accept or reject.

3.1 System Model

There are three entities in a naysayer proof system. We assume that all parties can read and write to a public bulletin board (e.g., a blockchain). Fix a function $f : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}$ and let \mathcal{L}_f be the language $\{(x, y) : \exists w \text{ s.t. } y = f(x, w)\}$. Let $\mathcal{R}_f = \{((x, y), w)\}$ be the corresponding relation. We assume f, x are known by all parties. When $f : \mathcal{Y} \times \mathcal{W} \to \mathcal{Y}$ is a state transition function with y' = f(y, w), this corresponds to the rollup scenarios described previously.

Prover The prover posts y and a proof π to the bulletin board claiming $(x, y) \in \mathcal{L}_f$.

- Verifier The verifier does not directly verify the validity of y or π , rather, it waits for time T_{nay} . If no one naysays (y,π) within that time, the verifier accepts y. In the pessimistic case, a party (or multiple parties) naysay the validity of π by posting proof(s) π_{nay} . The verifier checks the validity of each π_{nay} , and if any of them pass, it rejects y.
- Naysayer If Verify(crs, $(x, y), \pi) = 0$, then the naysayer posts a naysayer proof π_{nay} to the public bulletin board before T_{nay} time elapses.

Note that, due to the optimistic paradigm, we must assume a synchronous communication model: in partial synchrony or asynchrony, the adversary can arbitrarily delay the posting of naysayer proofs, and one cannot enforce soundness of the underlying proofs. Furthermore, we assume that the public bulletin board offers censorship-resistance, i.e., anyone who wishes to write to it can do so successfully within a certain time bound. Finally, we assume that there is at least one honest party who will submit a naysayer proof for any invalid π .

3.2 Formal Definitions

Next, we introduce a formal definition and syntax for naysayer proofs. A naysayer proof system Π_{nay} can be seen as a "wrapper" around an underlying proof system Π . For example, Π_{nay} defines a proving algorithm Π_{nay} . Prove which uses the original prover Π . Prove as a subroutine.

Definition 2 (Naysayer proof). Given a non-interactive proof system $\Pi = (\text{Setup}, \text{Prove}, \text{Verify})$ for an NP language \mathcal{L} , the corresponding naysayer proof system Π_{nay} is a tuple of PPT algorithms (Setup, Prove, Naysay, VerifyNay) defined as follows:

- Setup $(1^{\lambda}, 1^{\lambda_{nay}}) \rightarrow (crs, crs_{nay})$: Given (potentially different) security parameters 1^{λ} and $1^{\lambda_{nay}}$, output two common reference strings crs and crs_{nay}. This algorithm may use private randomness.
- Prove(crs, x, w) $\rightarrow \pi'$: Given a statement x and witness w such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$, output $\pi' = (\pi, \mathsf{aux})$, where $\pi \leftarrow \Pi$. Prove(crs, x, w).
- Naysay($\operatorname{crs}_{nay}, (x, \pi'), \operatorname{td}_{nay}$) $\to \pi_{nay}$: Given a statement x and values $\pi' = (\pi, \operatorname{aux})$ where π is a (potentially invalid) proof that $\exists w \text{ s.t. } (x, w) \in \mathcal{R}_{\mathcal{L}}$ using the proof system Π , output a naysayer proof π_{nay} disputing π . This algorithm may also make use of some (private) trapdoor information $\operatorname{td}_{nay} \subseteq w$.
- VerifyNay(crs_{nay}, $(x, \pi'), \pi_{nay}$) $\rightarrow \{0, \bot\}$: Given a statement-proof pair (x, π') and a naysayer proof π_{nay} disputing π' , output a bit indicating whether the evidence is sufficient to reject (0) or inconclusive (\bot) .

A trivial naysayer proof system always exists in which $\pi_{nay} = \top$, $\pi' = (\pi, \bot)$, and VerifyNay simply runs the original verification procedure, outputting 0 if Π .Verify(crs, $x, \pi) = 0$ and \bot otherwise. We say a proof system Π is *efficiently naysayable* if there exists a corresponding naysayer proof system Π_{nay} such that VerifyNay is asymptotically faster than Verify. If VerifyNay is only concretely faster than Verify, we say Π_{nay} is a *weakly efficient* naysayer proof. Note that some proof systems already have constant proof size and verification time [Gro16, Sch90] and therefore can, at best, admit only weakly efficient naysayer proofs. Moreover, if $td_{nay} = \bot$, we say Π_{nay} is a *public* naysayer proof (see Section 4.4 for an example of a non-public naysayer proof).

Definition 3 (Naysayer completeness). Given a proof system Π , a naysayer proof system $\Pi_{nay} = (\text{Setup}, \text{Prove}, \text{Naysay}, \text{VerifyNay})$ is complete if, for all honestly generated crs, crs_{nay} and all values of $\text{aux}, {}^4\text{given}$ an invalid statement-proof pair (x, π) , Naysay outputs a valid naysayer proof π_{nay} . That is, for all $\lambda, \lambda_{nay} \in \mathbb{N}$ and all aux, x, π ,

$$\Pr\left[\begin{array}{c} \operatorname{VerifyNay}(\operatorname{crs}_{\mathsf{nay}},(x,(\pi,\mathsf{aux})),\pi_{\mathsf{nay}})=0 \middle| \begin{array}{c} (\operatorname{crs},\operatorname{crs}_{\mathsf{nay}}) \leftarrow \operatorname{Setup}(1^{\lambda},1^{\lambda_{\mathsf{nay}}}) \wedge \\ \Pi.\operatorname{Verify}(\operatorname{crs},x,\pi) \neq 1 \wedge \\ \pi_{\mathsf{nay}} \leftarrow \operatorname{Naysay}(\operatorname{crs}_{\mathsf{nay}},(x,(\pi,\mathsf{aux})),\bot) \end{array} \right] = 1.$$

⁴We do not place any requirement on aux.

Definition 4 (Naysayer soundness). Given a proof system Π , a naysayer proof system Π_{nay} is sound if, for all PPT adversaries \mathcal{A} , and for all honestly generated crs, crs_{nay}, all $(x, w) \in \mathcal{R}_{\mathcal{L}}$, and all correct proofs π' , \mathcal{A} produces a passing naysayer proof π_{nay} with at most negligible probability. That is, for all λ , $\lambda_{nay} \in \mathbb{N}$, and all td_{nay},

$$\Pr\left[\operatorname{VerifyNay}(\operatorname{crs}_{\mathsf{nay}}, (x, \pi'), \pi_{\mathsf{nay}}) = 0 \middle| \begin{array}{c} (\operatorname{crs}, \operatorname{crs}_{\mathsf{nay}}) \leftarrow \operatorname{Setup}(1^{\lambda}, 1^{\lambda_{\mathsf{nay}}}) \land \\ (x, w) \in \mathcal{R}_{\mathcal{L}} \land \\ \pi' \leftarrow \operatorname{Prove}(\operatorname{crs}, x, w) \land \\ \pi_{\mathsf{nay}} \leftarrow \mathcal{A}(\operatorname{crs}_{\mathsf{nay}}, (x, \pi'), \operatorname{td}_{\mathsf{nay}}) \end{array} \right] \leq \operatorname{negl}(\lambda_{\mathsf{nay}}).^{5}$$

We distinguish between two types of naysayer proofs as follows.

- **Type 1.** A prover of an NP-relation $\mathcal{R}_{\mathcal{L}}$ posts (x, π) to the public bulletin board claiming that $x \in \mathcal{L}$. If the proof π is invalid with respect to the statement x, i.e., $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 0$, then naysayer provers convince the resourceconstrained verifier by sending a π_{nay} that this is indeed the case, i.e., $\mathsf{Verify}(\mathsf{ars}, \pi), \pi_{\mathsf{nay}} = 0$.
- **Type 2.** This family of naysayer proofs is even more efficient in the optimistic case, as the prover only sends the instance x and no proofs at all, claiming without evidence that $(x, w) \in \mathcal{R}_{\mathcal{L}}$. On the other hand, if the prover's assertion is incorrect, i.e., $(x, w) \notin \mathcal{R}_{\mathcal{L}}$, then a naysayer prover provides the correct statement x' such that $(x', w) \in \mathcal{R}_{\mathcal{L}}$ and a corresponding "regular" proof π such that $\operatorname{Verify}(\operatorname{crs}, x', \pi) = 1$. For example, in the case of rollups, the (public) witness w is the set of transactions in the rollup, and the statement $x = (\operatorname{st}, \operatorname{st}')$ is the updated rollup state after applying the batch w. Therefore, an incorrect assertion represents an incorrect application of the update w. The correction x' is the result of the proper application of w.

We conjecture that in most applications, in the worst case, type-2 naysayer proofs are more costly than type-1 naysayer proofs (both compute and storage). It is an interesting open question which applications are more suited to type-1 or type-2 naysayer proofs considering both optimistic and pessimistic costs. To thoroughly model this question, one must take into account the verifier's compute cost, the (naysayer) proof storage costs, as well as the probability of the prover sending an invalid proof. We leave this problem to future work. In the rest of this paper, we focus solely on type-1 naysayer proofs.

3.3 Naysayer Proofs for All NP

Next, we show that every proof system with polynomial-time verification has a corresponding naysayer proof system with a logarithmic-sized (in the size of the verification circuit) naysayer proofs and constant verification time—i.e., a succinct naysayer proof system.

Lemma 1. A claimed satisfying assignment for a circuit $C : \mathcal{X} \to \{0,1\}$ on input $x \in \mathcal{X}$ is efficiently naysayable. That is, if $C(x) \neq 1$, there is an $\mathcal{O}(\log |C|)$ -size proof of this fact which can be checked in constant-time, assuming oracle access to the wire assignments of C(x).

Proof. Without loss of generality, let C be a circuit of fan-in 2.

If $C(x) \neq 1$, then there must be some gate of C for which the wire assignment is inconsistent. Let i be the index of this gate (note $|i| \in \mathcal{O}(\log |C|)$). To confirm that $C(x) \neq 1$, a party can re-evaluate the indicated gate G_i on its inputs a, b and compare the result to the output wire c. That is, if $G_i(a, b) \neq c$, the verifier rejects the satisfying assignment.

Theorem 1. Every proof system Π with poly(|x|, |w|) verification complexity has a succinct naysayer proof.

Proof. Given any proof system Π , the evaluation of Π . Verify(crs, \cdot , \cdot) can be represented as a circuit C. (We assume this circuit description is public.) Then the following is a complete and sound naysayer proof system Π_{nay} :

Setup $(1^{\lambda}, 1^{\lambda_{nay}})$: Output crs $\leftarrow \Pi$.Setup (1^{λ}) and crs_{nay} := \emptyset .

Prove(crs, x, w) $\rightarrow \pi'$: Let $\pi \leftarrow \Pi$.Prove(crs, x, w) and aux be the wire assignments of Π .Verify(crs, x, π). Output $\pi' = (\pi, aux)$.

⁵If we assume aux is computed correctly, the second and third line of the precondition can be simplified to see that Π_{nay} is required to be a sound proof system for the language $\mathcal{L}_{nay} = \{(x, \pi) : x \notin \mathcal{L} \lor \Pi$. Verify(crs, $x, \pi) \neq 1$).

- Naysay(crs_{nay}, (x, π') , td_{nay}): Parse $\pi' = (\pi, aux)^6$ and output $\pi_{nay} := \top$ if aux = aux' || 0. Otherwise, evaluate Π .Verify(crs, x, π). If the result is not 1, search aux to find an incorrect wire assignment for some gate $G_i \in C$. Output $\pi_{nay} := i$.
- VerifyNay(crs_{nay}, $(x, \pi'), \pi_{nay}$): Parse $\pi' = (\cdot, aux)$ and $\pi_{nay} = i$. If $aux = aux' \parallel 0$, output 0, indicating rejection of the proof π' . Otherwise, obtain the values in, $out \in aux$ corresponding to the gate G_i and check $G_i(in) \stackrel{?}{=} out$. If the equality does not hold, output \bot ; else output 0.

Completeness (if a π fails to verify, we can naysay (π, aux)) follows by Lemma 1. If Π .Verify(crs, $x, \pi) \neq 1$, then we have two cases: If aux is consistent with a correct evaluation of Π .Verify(crs, x, π), either aux = aux' || 0 (and VerifyNay rejects) or we can apply the lemma to find an index i such that $G_i(in) \neq out$ for in, $out \in aux$, where $G_i \in C$. Alternatively, if aux is not consistent with a correct evaluation, there must be some gate (with index i') which was evaluated incorrectly, i.e., $G_{i'}(in) \neq out$ for in, $out \in aux$.

Soundness follows by the completeness of Π . If $(x, w) \in \mathcal{R}_{\mathcal{L}}$ and $\pi' = (\pi, \mathsf{aux})$ is computed correctly, completeness of Π implies Π . Verify(crs, $x, \pi) = 1$. Since aux is correct, it follows that $\mathsf{aux} \neq \mathsf{aux} || 0$ and $G_i(\mathsf{in}) = \mathsf{out}$ for all $i \in |C|$ and corresponding values $\mathsf{in}, \mathsf{out} \in \mathsf{aux}$. Thus there is no index i which will cause $\mathsf{VerifyNay}(\mathsf{crs}_{\mathsf{nay}}, (x, \pi'), i)$ to output 0.

Succinctness of π_{nay} follows from the fact that $|i| = \log |\Pi.\text{Verify}(crs, \cdot, \cdot)| = \mathcal{O}(\log(|x|, |w|)) \in o(|x| + |w|)$ and that the runtime of VerifyNay is constant.

The proof of Theorem 1 gives a generic way to build a succinct naysayer proof system for any proof system Π with polynomial-time verification. For succinct proof systems, the generic construction even allows efficient (sublinear) naysaying, since the runtime of Naysay depends only on the runtime of Π . Verify, which is sublinear if Π is succinct.

Notice that although the syntax gives $\pi' = (\pi, aux)$ as an input to the VerifyNay algorithm, in the generic construction the algorithm does not make use of π . Thus, if a naysayer rollup were instantiated with this generic construction, π would not need to be posted on-chain since the on-chain verifier (running the VerifyNay algorithm) will not use this information. In fact, the verifier wouldn't even need most of aux—only the values corresponding to the gate G_i , which is determined by π_{nay} . Thus, although π' must be available to all potential naysayers, only a small (adaptive) fraction of it must be accessible on-chain. In Section 5, we will discuss how to leverage this insight to reduce the storage costs of a naysayer rollup.

4 Four Concrete Applications of Naysayer Proofs

The naysayer proof paradigm is generally applicable for proof systems with multi-round amplification, repetitive structure (e.g., multiple bilinear pairing checks [GWC19]), or recursive reduction (e.g., Pietrzak's proof of exponentiation [Pie19]). In this section, we highlight four example constructions of naysayer proofs. Our first construction (Section 4.1) is a concrete example of the generic naysayer construction from Theorem 1, applied to Merkle trees. We then consider two cases which take advantage of repetition in the verification procedure to achieve better naysayer performance: the FRI polynomial commitment scheme (Section 4.2) and two post-quantum signature schemes (Section 4.3). Then, in Section 4.4, we give an example of a non-public naysayer proof which uses a trapdoor to reduce the size and verification complexity of the naysayer proof.

4.1 Merkle Commitments

	Proof size	Verification
Original	$\log n \ \mathbb{H}$	$\log n \mathbb{H}$
Naysayer	$\log \log n \mathbb{B}$	$1\mathbb{H}$

Table 2: Cost savings of the naysayer paradigm applied to Merkle proofs. \mathbb{H} = hash output size/hash operations, \mathbb{B} = bits.

Merkle trees [Mer88] and their variants are ubiquitous in modern systems, including Ethereum's state storage [Eth24]. A Merkle tree can be used to commit to a vector \vec{v} of elements as shown in Figure 1, with the root h

⁶If |aux| is larger than the number of wires in C, truncate it to the appropriate length.



Figure 1: Each node in a Merkle tree consists of a hash of its children. The root h is a commitment to the vector of leaves (v_0, v_1, \ldots, v_7) . An opening proof for the element v_2 is its copath (black nodes); the "verification trace" for the proof is the path (gray nodes).

acting as a commitment to \vec{v} . The party who created the tree can prove the inclusion of some element v_i at position i in the tree by providing the corresponding copath.

For example, to open the leaf at position 2, a prover provides its value v_2 and an opening proof $\pi = (h_{011}, h_{00}, h_1)$ consisting of the copath from the leaf v_2 to the root h. The proof π is checked by using its contents to recompute the root h' starting with v_2 , then checking that h = h'. This involves recomputing the nodes along the path from the leaf to the root (the gray nodes in the figure). These nodes can be seen as a "verification trace" for the proof π .

In the context of a naysayer proof system, the prover provides π along with the vertication trace $aux = (h_{010}, h_{01}, h_0)$. A naysayer can point out an error at a particular point of the trace by submitting the incorrect index of aux (e.g., $\pi_{nay} = 1$ to indicate h_{01}). The naysayer verifier checks π_{nay} by computing a single hash using π and oracle access to aux, e.g., checking $H(h_{010}, h_{011}) \stackrel{?}{=} h_{01}$, where $h_{010}, h_{01} \in aux$ and $h_{011} \in \pi$. This is the generic construction from Theorem 1.

4.2 FRI Polynomial Commitment Scheme

	Proof size	Verification
Original	$\mathcal{O}(\lambda \log^2 d)\mathbb{H} + \mathcal{O}(\lambda \log d)\mathbb{F}$	$\mathcal{O}(\lambda \log^2 d)\mathbb{H} + \mathcal{O}(\lambda \log d)\mathbb{F}$
Naysayer	$2\log(q\log d) + 1 \mathbb{B}$	best: $\mathcal{O}(1)\mathbb{F}$ worst: $\mathcal{O}(\log d)\mathbb{H}$

Table 3: Cost savings of the naysayer paradigm applied to FRI opening proofs. \mathbb{H} = hash output size/hash operations, \mathbb{F} = field element size/operations, \mathbb{B} = bits.

The Fast Reed-Solomon IOP of proximity (FRI) [BBHR18a] is used as a building block in many non-interactive proof systems, including the STARK IOP [BBHR18b]. Below, we describe only the parts of FRI as applied in STARK. We refer the reader to the cited works for details.

The FRI commitment to a polynomial $p(X) \in \mathbb{F}[X]^{\leq d}$ is the root of a Merkle tree with $\rho^{-1}d$ leaves. Each leaf is an evaluation of p(X) on the set $L_0 \subset \mathbb{F}$, where $\rho^{-1}d = |L_0| \ll |\mathbb{F}|$ for a constant $0 < \rho < 1$ (the Reed-Solomon rate parameter). We focus on the verifier's cost in the proof of proximity. Let δ be a parameter of the scheme such that $\delta \in (0, 1 - \sqrt{\rho})$. The prover sends $\log d + 1$ values (roots of successive "foldings" of the original Merkle tree, plus the value of the constant polynomial encoded by the final tree). The verifier makes $q = \lambda/\log(1/(1-\delta))$ queries to ensure $2^{-\lambda}$ soundness error; the prover responds to each query with $2 \log d$ Merkle opening proofs (2 for each folded root). For each query, the verifier must check each Merkle authentication path, amounting to $\mathcal{O}(\log d \log \rho^{-1}d)$ hashes per query. Furthermore, it must perform $\log d$ arithmetic checks (roughly 3 additions, 2 divisions, and 2 multiplications in \mathbb{F} per folding) per query to ensure the consistency of the folded evaluations. Therefore, the overall FRI verification consists of $\mathcal{O}(\lambda \log^2 d)$ hashes and $\mathcal{O}(\lambda \log d)$ field operations. A FRI proof is invalid if any of the above checks fails. Therefore a straightforward naysayer proof $\pi_{nay}^{\mathsf{FRI}} = (i, j, k)$ need only point out a single Merkle proof (the *j*th proof for the *i*th query, $i \in [q], j \in [2 \log d]$) or a single arithmetic check $k \in [q \log d]$ which fails. The naysayer verifier only needs to recompute that particular check: $\mathcal{O}(\log \rho^{-1}d)$ hashes in the former case⁷ or a few arithmetic operations over \mathbb{F} in the latter.

This approach can lead to incredible concrete savings: According to [Hab22], for $\lambda = 128$, $d = 2^{12}$, $\beta \rho = 2^{-3}$, q = 91, $\delta = 9$, the size of a vanilla FRI opening proof (i.e., without concrete optimizations) can be estimated at around 322KB. A naysayer proof for the same parameter settings is $2 \log(q \log d) + 1 \approx 2 \cdot 10 + 1 = 21$ bits ; 3 bytes.

4.3 Post-quantum Signature Schemes

	Proof size	Verification
Original	$\mathcal{O}(\lambda)\mathbb{F}$	$\mathcal{O}(\lambda)\mathbb{F} + 1\mathbb{H}$
Naysayer	$2 + \log k + \log d \mathbb{B}$	best: $\mathcal{O}(1)\mathbb{F}$ worst: $\mathcal{O}(\lambda)\mathbb{F} + 1\mathbb{H}$

Table 4: Cost savings of the naysayer paradigm applied to CRYSTALS-Dilithium signatures. \mathbb{H} = hash output size/hash operations, \mathbb{F} = field element size/operations, \mathbb{B} = bits. Since the parameter k depends on λ and d is a constant, $|\pi_{nay}| \in \mathcal{O}(\log \lambda)$.

With the advent of account abstraction [Eth23a], Ethereum users can define their own preferred digital signature schemes, including post-quantum signatures as recently standardized by NIST [BHK⁺19, DKL⁺18, PFH⁺22]. Compared to their classical counterparts, post-quantum signatures generally have either substantially larger signatures or substantially larger public keys. Since this makes post-quantum signatures expensive to verify on-chain, these schemes are prime candidates for the naysayer proof paradigm.

CRYSTALS-Dilithium [**DKL**⁺18]. We give a simplified version of signature verification in lattice-based signatures like CRYSTALS-Dilithium. In these schemes, the verifier checks that the following holds for a signature $\sigma = (\vec{z_1}, \vec{z_2}, c)$, public key $pk = (\vec{A}, \vec{t})$, and message M:

$$\|\vec{z}_1\|_{\infty} < \beta \land \|\vec{z}_2\|_{\infty} < \beta \land c = H(M, \vec{w}, pk).$$

$$\tag{1}$$

Here β is a constant, $\vec{A} \in R_q^{k \times \ell}$, $\vec{z}_1 \in R_q^{\ell}$, $\vec{z}_2, \vec{t} \in R_q^k$ for the polynomial ring $R_q := \mathbb{Z}_q[X]/(X^d + 1)$, and $\vec{w} = \vec{A}\vec{z}_1 + \vec{z}_2 - c\vec{t} \mod q$. (Dilithium uses d = 256.) We will write elements of R_q as polynomials $p(X) = \sum_{j \in [d]} \alpha_j X^j$ with coefficients $\alpha_j \in \mathbb{Z}_q$. Since Equation (1) is a conjunction, the naysayer prover must show that

$$(\exists z_i \in \vec{z}_1, \vec{z}_2 : \|z_i\|_{\infty} > \beta) \lor c \neq H(M, \vec{w}, pk).$$

$$\tag{2}$$

If the first check of Equation (1) fails, the naysayer gives an index *i* for which the infinity norm of one of the polynomials in $\vec{z_1}$ or $\vec{z_2}$ is large. (In particular, it can give a tuple (b, i, j) such that $\alpha_j > \beta$ for $z_i = \cdots + \alpha_j X^j + \ldots \in \vec{z_b}$.)⁹

If the second check fails, the naysayer indicates that clause to the naysayer verifier, who must recompute \vec{w} and perform a single hash evaluation which is compared to c.

Overall, π_{nay} is a tuple (a, b, i, j) indicating a clause $a \in [2]$ of Equation (2), the vector \vec{z}_b with $b \in [2]$, an entry $i \in [\max\{k, \ell\}]$ in that vector, and the index $j \in [d]$ of the offending coefficient in that entry. Since $k \ge \ell$, we have $|\pi_{nay}| = (2 + \log k + \log d)$ bits. The verifier is very efficient when naysaying the first clause, and only slightly faster than the original verifier for the second clause.

SPHINCS+ [**BHK**⁺19]. The signature verifier in SPHINCS+ checks several Merkle authentication proofs, requiring hundreds or even thousands of hash evaluations. An efficient naysayer proof can be easily devised akin to the Merkle naysayer described in Section 4.1. Given a verification trace, the naysayer prover simply points to the hash evaluation in one of the Merkle-trees where the signature verification fails.

 $^{^{7}}$ One could use a Merkle naysayer proof (Section 4.1) to further reduce the naysayer verification from checking a full Merkle path to a single hash evaluation.

⁸This is smaller than most polynomial degrees used in production systems today.

⁹The same idea can be applied to constructions bounding the ℓ_2 norm, but with lower efficiency gains for the naysayer verifier, who must recompute the full ℓ_2 norm of either \vec{z}_1, \vec{z}_2 .

4.4 Verifiable Shuffles

	Proof size	Verification
Original	$\mathcal{O}(\sqrt{n})\mathbb{G}$	$\mathcal{O}(n)\mathbb{G}$
Naysayer	$\log n \mathbb{B} + 3\mathbb{G} + 1\mathbb{F}$	$\mathcal{O}(1)\mathbb{G} + 1\mathbb{H}$

Table 5: Cost savings of the naysayer paradigm applied to Bayer-Groth shuffles. \mathbb{H} = hash output size/hash operations, \mathbb{G} = group element size/operations, \mathbb{B} = bits.

Verifiable shuffles are applied in many (blockchain) applications such as single secret leader election algorithms [BEHG20], mix-nets [Cha81], cryptocurrency mixers [SNBB19], and e-voting [Adi08]. The state-of-the-art proof system for proving the correctness of a shuffle is due to Bayer and Groth [BG12]. Their proof system is computationally heavy to verify on-chain as the proof size is $\mathcal{O}(\sqrt{n})$ and verification time is $\mathcal{O}(n)$, where n is the number of shuffled elements.

Most shuffling protocols (of public keys, re-randomizable commitments, or ElGamal ciphertexts) admit a particularly efficient naysayer proof if the naysayer knows at least one of the shuffled elements. Let us consider the simple case of shuffling public keys. The shuffler wishes to prove membership in the following NP language:

$$\mathcal{L}_{perm} := \{ ((pk_i, pk'_i)_{i=1}^n, R) : \exists r, w_1, \dots, w_n \in \mathbb{F}_p, \sigma \in \mathsf{Perm}(n) \text{ s.t. } \forall i \in [n], pk_i = g^{w_i} \land pk'_i = g^{r \cdot w_{\sigma(i)}} \land R = g^r \}.$$

$$(3)$$

Here $\mathsf{Perm}(n)$ is the set of all permutations $f: [n] \to [n]$.

Suppose a party knows that for some $j \in [n]$, the prover did not correctly include $pk'_j = g^{r \cdot w_j}$ in the shuffle. The party can naysay by showing that

$$(g, pk_j, R, pk'_i) \in \mathcal{L}_{DH} \land pk'_i \notin (pk_i, \cdot)_{i=1}^n$$

where \mathcal{L}_{DH} is the language of Diffie-Hellman tuples¹⁰. To produce such a proof, however, the naysayer must know the discrete logarithm w_j . Unlike our previous examples, which were public naysayer proofs, this is an example of a private Naysay algorithm using $\mathsf{td}_{\mathsf{nay}} := w_j$. The naysayer proof is $\pi_{\mathsf{nay}} := (j, pk'_j, \pi_{DH})$. The Diffie-Hellman proof can be checked in constant time and, with the right data structure for the permuted list (e.g., a hash table), so can the list non-membership. This, π_{nay} is a $\mathcal{O}(\log n)$ -sized naysayer proof with $\mathcal{O}(1)$ -verification, yielding in exponential savings compared to verifying the original Bayer-Groth shuffle proof.

4.5 Summary

We showed the asymptotic cost savings of the verifiers in the four examples discussed in Sections 4.1 to 4.4 in their respective tables. Note that the verifier speedup is exponential for verifiable shuffles and logarithmic for the Merkle and FRI openings. For CRYSTALS-Dilithium, our naysayer proof is only *weakly efficient* (see Section 3.2) as there is no asymptotic gap in the complexity of the original signature verification and the naysayer verification in the worst case.

As for proof size, in all the examples, our naysayer proofs are logarithmically smaller than the original proofs. (Note this calculation does not include the size of aux, but we will see in the next section that aux does not meaningfully impact the proof size for the verifier.) Furthermore, in most cases, the naysayer proof consists of an *integer* index or indices rather than group or field elements. Representing the former requires only a few bits compared to the latter (which are normally at least λ bits long), so in practice, naysayer proofs can offer *practically* smaller proofs sizes even when they are not asymptotically smaller. This can lead to savings even when the original proof is constant-size (e.g., a few group elements).

5 Storage Considerations

We assumed in our evaluation that the naysayer verifier can read the instance x, the original proof π , and the naysayer proof π_{nay} entirely. Note that in the pessimistic case, the verifier requires increased storage (for π_{nay}) but

¹⁰Membership in \mathcal{L}_{DH} can be shown via a proof of knowledge of discrete logarithm equality [CP93] consisting of 2 group elements and 1 field element which can be verified with 4 exponentiations and 2 multiplications in the group.

only needs to compute VerifyNay instead of Verify. A useful naysayer proof system should compensate for increased storage by considerably reducing verification costs.

In either case, this approach of storing all data on chain may not be sufficient in blockchain contexts where storage is typically very costly. Blockchains such as Ethereum differentiate costs between persistent storage (which we can call S_{per}) and "call data" (S_{call}), which is available only for one transaction and is significantly cheaper as a result. Verifiable computation proofs, for example, are usually stored in S_{call} with only the verification result persisted to S_{per} .

Some applications now use a third, even cheaper, tier of data storage, namely off-chain data availability services (S_{DA}) , which promise to make data available off-chain but which on-chain contracts have no ability to read. Verifiable storage, an analog of verifiable computation, enables a verifier to store only a short commitment to a large vector [CF13, Mer88] or polynomial [KZG10], with an untrusted storage provider (S_{DA}) storing the full values. Individual data items (elements in a vector or evaluations of the polynomial) can be provided as needed to S_{call} or S_{per} with short proofs that they are correct with respect to the stored commitment. (Ethereum implemented this type of storage, commonly referred to as "blob data", using KZG commitments in EIP-4844 [BFL+22].)

This suggests an optimization for naysayer proofs in a blockchain context: the prover posts only a binding commitment $\text{Com}(\pi')$, which the contract stores in S_{per} , while the actual proof $\pi' = (\pi, aux)$ is stored in S_{DA} . We assume that potential naysayers can read π' from S_{DA} . In the optimistic case, the full proof π' is never written to the more-expensive S_{call} or S_{per} . In the pessimistic case, when naysaying is necessary, the naysayer must send openings of the erroneous proof elements to the verifier (in S_{call}). The verifier checks that these data elements are valid with respect to the on-chain commitment $\text{Com}(\pi')$ stored in S_{per} . Note that most naysayer proof systems don't require reading all of π' for verification, so even this pessimistic case will offer significant savings over storing all of π' in S_{call} . An important future research direction is to investigate this optimized storage model's implications and implementation details.

6 Open Questions and Conclusion

We see many exciting open research directions for naysayer proofs. A thorough game-theoretical analysis of naysayer proofs (e.g., deposits and the length of the challenge period) is crucial for real-world deployments. Another fascinating direction is to better understand the complexity-theoretic properties of naysayer proofs. Is it possible to create a universal black-box naysayer proof for all non-interactive proof systems? Finally, one might consider several extensions of naysayer proofs, e.g., interactive naysayer proofs or naysayer proofs with non-negligible soundness error. We leave these generalizations to future work.

Acknowledgements. We thank Miranda Christ, Mahimna Kelkar, Joachim Neu, Valeria Nikolaenko, Ron Rothblum, Ertem Nusret Tas, and Justin Thaler for insightful discussions. We also thank Wei Dai and Massimiliano Sala for pointing out an issue in our main theorem which has since been fixed. Alex Block and Doruk Gür also gave helpful feedback on the naysayer constructions for FRI and Dilithium, respectively. This work was supported by al6z crypto research. István András Seres was partially supported by the Ministry of Culture and Innovation and the National Research, Development, and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004). Joseph Bonneau was additionally supported by DARPA Agreement and NSF grant CNS-2239975. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, DARPA, al6z, or any other supporting organization.

References

- [AAB⁺24] Mario M Alvarez, Henry Arneson, Ben Berger, Lee Bousfield, Chris Buckland, Yafah Edelman, Edward W Felten, Daniel Goldman, Raul Jordan, Mahimna Kelkar, et al. Bold: Fast and cheap dispute resolution. arXiv preprint arXiv:2404.10491, 2024. [page 2.]
- [Adi08] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, USENIX Security 2008, pages 335–348. USENIX Association, July / August 2008. [page 9.]
- [BBHR18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald

Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018. [page 7.]

- [BBHR18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and postquantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https: //eprint.iacr.org/2018/046. [page 7.]
- [BEHG20] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In Advances in Financial Technologies, 2020. [page 9.]
- [BFL⁺22] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. Eip-4844: Shard blob transactions. https://eips.ethereum.org/EIPS/eip-4844, Feb 2022. [page 10.]
- [BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT 2012, volume 7237 of LNCS, pages 263–280. Springer, Heidelberg, April 2012. [page 9.]
- [BHK⁺19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS⁺ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, ACM CCS 2019, pages 2129–2146. ACM Press, November 2019. [page 8.]
- [Buc21] Chris Buckland. Fraud proofs and virtual machines. https://medium.com/@cpbuckland88/ fraud-proofs-and-virtual-machines-2826a3412099, October 2021. [page 2.]
- [CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013. [page 10.]
- [Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications* of the ACM, 24(2), 1981. [page 9.]
- [CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, CRYPTO'92, volume 740 of LNCS, pages 89–105. Springer, Heidelberg, August 1993. [page 9.]
- [DKL⁺18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839. [page 8.]
- [Eth23a] Ethereum. Account Abstraction. ethereum.org/en/roadmap/account-abstraction, 2023. [page 8.]
- [Eth23b] Ethereum. Optimistic rollups. https://ethereum.org/en/developers/docs/scaling/ optimistic-rollups/, 2023. [page 2.]
- [Eth24] Ethereum. Merkle patricia trie. https://ethereum.org/en/developers/docs/ data-structures-and-encoding/patricia-merkle-trie/, jul 2024. [page 6.]
- [Fic] Kelvin Fichter. Why is the optimistic rollup challenge period 7 days? https://kelvinfichter.com/ pages/thoughts/challenge-periods/. [page 2.]
- [FK97] Uriel Feige and Joe Kilian. Making games short (extended abstract). In 29th ACM STOC, pages 506–516. ACM Press, May 1997. [page 2.]
- [GGP10] 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, Heidelberg, August 2010. [page 1.]
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, EUROCRYPT 2016, Part II, volume 9666 of LNCS, pages 305–326. Springer, Heidelberg, May 2016. [page 4.]

- [GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrangebases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953. [page 6.]
- [Hab22] Ulrich Haböck. A summary on the FRI low degree test. Cryptology ePrint Archive, Report 2022/1216, 2022. https://eprint.iacr.org/2022/1216. [page 8.]
- [KGC⁺18] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In William Enck and Adrienne Porter Felt, editors, USENIX Security 2018, pages 1353–1370. USENIX Association, August 2018. [pages 2 and 3.]
- [KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, ASIACRYPT 2010, volume 6477 of LNCS, pages 177–194. Springer, Heidelberg, December 2010. [page 10.]
- [Lab23] Offchain Labs. Inside Arbitrum Nitro. https://docs.arbitrum.io/inside-arbitrum-nitro/, 2023. [page 2.]
- [Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988. [pages 6 and 10.]
- [Opt23] Optimism. Rollup Protocol. https://community.optimism.io/docs/protocol/ 2-rollup-protocol, 2023. [page 2.]
- [PB17] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts, 2017. [page 3.]
- [PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2016. [page 3.]
- [PFH⁺22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/ Projects/post-quantum-cryptography/selected-algorithms-2022. /page 8.]
- [Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019. [page 6.]
- [Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990. [page 4.]
- [Sin 22]Sritanshu Sinha. Can the Optimism blockchain the battle of the win rollups? Cointelegraph, June 2022.https://cointelegraph.com/news/ can-the-optimism-blockchain-win-the-battle-of-the-rollups. [page 2.]
- [SJSW19] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. ETHDKG: Distributed key generation with Ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985, 2019. https://eprint.iacr.org/2019/985. [page 3.]
- [SNBB19] István András Seres, Dániel A. Nagy, Chris Buckland, and Péter Burcsi. MixEth: efficient, trustless coin mixing service for Ethereum. Cryptology ePrint Archive, Report 2019/341, 2019. https:// eprint.iacr.org/2019/341. [pages 3 and 9.]
- [Tha23] Justin Thaler. Proofs, arguments, and zero-knowledge. https://people.cs.georgetown.edu/ jthaler/ProofsArgsAndZK.pdf, July 2023. [page 3.]
- [TR19] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. arXiv preprint arXiv:1908.04756, 2019. [page 2.]
- [W⁺14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014. [page 1.]