Naysayer proofs

István András Seres[1], Noemi Glaeser[2], and Joseph Bonneau[3]

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

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 constant-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 [32], 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* [16], 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 (eq. 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.[4]

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

[4] https://etherscan.io/address/0x3e6118da317f7a433031f03bb71ab870d87dd2dd.

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.[5]

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 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.

2 Related work

A concept related to the naysayer paradigm is *refereed delegation* [14]. The idea has found widespread adoption in the form of "fraud proofs" or "fault proofs" as used in *optimistic rollups* [13,19,22,30]. 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 case of a challenge, a dispute resolution process ensues between the *challenger* and the *defender*, which can be either non-interactive or interactive. In the former approach, the full computation is re-executed by the on-chain verifier to resolve the dispute. In the latter approach, the challenger and defender engage in a *bisection protocol* to locate a disputed step of the computation, and only that step is re-executed to resolve the dispute.

At a high level, in the fraud proof paradigm, a "prover" performs a provisionally accepted computation without any proof of correctness. Any party can then challenge the correctness of the prover's *computation*. In the naysayer

[5] At the time of this writing, we are unaware of any major rollup service which has posted an incorrect proof in production.

paradigm, by contrast, the prover supplies a proof with the computation output, which is provisionally accepted. Any party can then challenge the correctness of *the proof*. The naysayer approach offers significant speedups since the verifier's circuit is typically much smaller than the original computation. Note that there is a slight semantic difference: fraud proofs can definitively show that the computation output is incorrect. In contrast, naysayer proofs can only show that the accompanying proof is invalid—the computation itself may have been correct.

Furthermore, for fraud proofs, the full computation input (the witness) must be made available to the verifier and potential challengers. Naysayer proofs, on the other hand, can be verified using only the statement and proof. Hence, naysayer proofs work naturally with zero-knowledge proofs. This can also lead to crucial savings if the witness is very large (e.g., transaction data for a rollup).

The fraud proof design pattern has been applied in an application-specific way in many blockchain applications besides optimistic rollups, including the Lightning Network [25], Plasma [24], cryptocurrency mixers [29], and distributed key generation [27]. We view naysayer proofs as a drop-in replacement for the many application-specific fault proofs, which are both more general and efficient.

3 Naysayer proofs

In this section, we introduce our system model and the syntax of naysayer proofs and show that constant-size and constant-time verifiable (i.e., succinct) naysayer proofs exist for all NP languages. First, we recall the syntax of non-interactive (zero-knowledge) proofs. We refer the reader to [31] 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 relation* $\mathcal{R}$ *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**:** *Given a security parameter, output a common reference string* crs*. This algorithm might use private randomness (a trusted setup).*

$\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$**:** *Given the* crs*, an instance* x*, and witness* w *such that* $(x, w) \in \mathcal{R}$*, output a proof* π*.*

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0, 1\}$**:** *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).

Prover The prover posts a proof π to the bulletin board claiming $(x, w) \in \mathcal{R}$.

Verifier The verifier does not directly verify the validity of π, rather, it allows everyone to naysay in a pre-defined time window of duration T_{nay}. Optimistically, if no one naysays π within time T_{nay}, the verifier accepts it. 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 the original proof π.

Naysayer If $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 0$, then the naysayer posts a naysayer proof π_{nay} to the public bulletin board before T_{nay} time elapses.

Note that we need to assume a synchronous communication model as we cannot have naysayer proofs in partial synchrony or asynchrony. If the adversary can arbitrarily delay the posting of naysayer proofs, then we cannot have sound proof systems. Note that synchrony is already assumed by most of the deployed consensus algorithms, e.g., Nakamoto consensus [21]. Furthermore, we assume that the public bulletin board offers censorship resistance for the writers of the bulletin board. Finally, we assume that there is *at least one honest party* who is ready to create and submit naysayer proofs for invalid proofs.

3.2 Naysayer proof system definitions and security

We formally introduce the notion of a *naysayer proof*, with the following syntax:

Definition 2 (Naysayer proof). *Given a non-interactive proof system* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$*, the corresponding naysayer proof system* Π_{nay} *is a tuple of PPT algorithms* $(\mathsf{NSetup}, \mathsf{Naysay}, \mathsf{VerifyNay})$ *defined as follows:*

$\mathsf{NSetup}(1^{\lambda_{\text{nay}}}) \to \mathsf{crs}_{\text{nay}}$**:** *Given a security parameter* λ_{nay}*, output a common reference string* $\mathsf{crs}_{\text{nay}}$*. This algorithm might use private randomness.*

$\mathsf{Naysay}(\mathsf{crs}_{\text{nay}}, (x, \pi), \mathsf{aux}_{\text{nay}}) \to \pi_{\text{nay}}$**:** *Given a statement* x*, a corresponding (potentially invalid) proof* π *in proof system* Π*, and auxiliary information* $\mathsf{aux}_{\text{nay}}$*, output a naysayer proof* π_{nay} *disputing* π*.*

$\mathsf{VerifyNay}(\mathsf{crs}_{\text{nay}}, (x, \pi), \pi_{\text{nay}}) \to \{0, \perp\}$**:** *Given a statement-proof pair* (x, π) *and a naysayer proof* π_{nay} *disputing* π*, output a bit indicating whether the evidence against* π *is sufficient to reject* π *(0) or inconclusive* ($\perp$)*.*

A trivial naysayer proof system always exists in which $\pi_{\text{nay}} = \varnothing$ and $\mathsf{VerifyNay}$ simply runs the original verification procedure. We say a proof system Π is *efficiently naysayable* if there exists a corresponding naysayer proof system Π_{nay} such that $\mathsf{VerifyNay}$ is asymptotically faster than Verify. If $\mathsf{VerifyNay}$ 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 [17,28] and therefore can, at best, admit weakly efficient naysayer proofs. Moreover, if $\mathsf{aux}_{\text{nay}} = \varnothing$, we say Π_{nay} is a *public* naysayer proof.

Definition 3 (Naysayer correctness). *Given a proof system* Π*, a naysayer proof system* Π_{nay} *is* correct *if, for all* $\mathsf{aux}_{\text{nay}}, \mathsf{crs}, x$*, and invalid proofs* π*,* Naysay *outputs a valid naysayer proof* π_{nay}*:*

$$\Pr\left[\mathsf{VerifyNay}(\mathsf{crs}_{\text{nay}}, (x, \pi), \pi_{\text{nay}}) = 0 \;\middle|\; \begin{array}{c} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 0 \;\wedge \\ \mathsf{crs}_{\text{nay}} \leftarrow \mathsf{NSetup}(1^{\lambda}) \;\wedge \\ \pi_{\text{nay}} \leftarrow \mathsf{Naysay}(\mathsf{crs}_{\text{nay}}, (x, \pi), \mathsf{aux}_{\text{nay}}) \end{array}\right] = 1. \tag{1}$$

Definition 4 (Naysayer soundness). *Given a proof system* Π*, a naysayer proof system* Π_{nay} *is sound if,* $\forall$*PPT adversaries* $\mathcal{A}$ *and* $\forall \mathsf{aux}, \mathsf{crs}, x$*, and correct proofs* π*,* $\mathcal{A}$ *produces a verifying naysayer proof* π_{nay} *with negligible probability:*

$$\Pr\left[\mathsf{VerifyNay}(\mathsf{crs}_{\text{nay}}, (x, \pi), \pi_{\text{nay}}) = 0 \left| \begin{array}{c} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge \\ \mathsf{crs}_{\text{nay}} \leftarrow \mathsf{NSetup}(1^\lambda) \wedge \\ \pi_{\text{nay}} \leftarrow \mathcal{A}(\mathsf{crs}_{\text{nay}}, (x, \pi), \mathsf{aux}_{\text{nay}}) \end{array} \right.\right] \leq \mathsf{negl}(\lambda). \tag{2}$$

3.3 Naysayer proofs for all NP

Finally, we show that for every NP language, there exists a constant-size naysayer proof with constant verification time (i.e., a succinct naysayer proof).

Theorem 1. *For every NP language* $\mathcal{L}$ *with relation* $\mathcal{R}_{\mathcal{L}}$*, there exists a naysayer proof system* Π_{nay} *with constant-size proof* π_{nay} *and constant-time verifier.*

Proof. By the Cook-Levin theorem [10], any NP language has a corresponding boolean circuit C such that $C(x) = 1$ if and only if $x \in \mathcal{L}$ (i.e., circuit satisfiability or SAT). A satisfying wire assignment w is, therefore, a (neither zero-knowledge nor succinct) proof that $x \in \mathcal{L}$ which admits constant-size and constant-time naysaying: if the wire assignment w is incorrect, there must be some gate of the circuit for which the wire assignment is inconsistent. The naysayer simply provides the index of this gate; the verifier then checks if the relevant wire assignments are consistent with a correct evaluation of the gate, which is a constant-time operation. Furthermore, the naysayer proof consists of a single element (the gate index), so it is constant-sized, i.e., succinct.

Corollary 1. *Every efficient proof system* Π *(i.e., with a polynomial-time verification algorithm) has a succinct naysayer proof.*

Proof. Given any proof system Π, one can represent the $\mathsf{Verify}(\mathsf{crs}, \cdot, \cdot)$ algorithm as a circuit and apply the above theorem to obtain a succinct naysayer proof.

4 Three 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 [15]), or recursive reduction (e.g., Pietrzak's proof of exponentiation [23]). In this section, we highlight three example constructions of naysayer proofs.

4.1 FRI polynomial commitment scheme

The FRI polynomial commitment scheme [3] is used as a building block in many non-interactive proof systems, including STARKs [4]. Below, we describe only the parts of FRI relevant to our discussion. The FRI commitment to a polynomial $p(x) \in \mathbb{F}^{\leq d}[X]$ 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 $0 < \rho < 1$. We focus on the verifier's cost in the opening proof of the FRI polynomial commitment scheme as applied in the STARK IOP. Let δ be a parameter of the scheme such that $\delta \in (0, 1 - \sqrt{\rho})$. The prover sends the verifier $\log_2(|L_0|)$ messages. The FRI opening proof's verifier queries the prover's each message $\lambda/\log_2(1/(1-\delta))$ times to ensure $2^{-\lambda}$ soundness error. In each query, the verifier needs to check a Merkle-tree authentication path consisting of $\mathcal{O}(\log_2(\rho^{-1}d))$ hashes. Therefore, the overall STARK proof consists of $\mathcal{O}(\lambda \log_2(\rho^{-1}d)/\log_2(1/(1-\delta))$ hashes.

The overall STARK proof is invalid if any of the individual Merkle proofs is invalid. Therefore a straightforward naysayer proof $\pi_{\mathsf{nay}}^{\mathsf{FRI}} = (i, z_i)$ need only point to the ith node in one of the Merkle proofs, where the hash values of the children nodes x, y and their parent node $z \neq H(x, y)$ do not match in one of the incorrect Merkle authentication paths. The naysayer verifier only needs to compute a single hash evaluation $H(x, y) = z_i$ and check $z_i \neq z$. Thus, the naysayer proof for FRI has constant-size and can be verified in constant-time.

4.2 Post-quantum signature schemes

With the advent of account abstraction [12], Ethereum users can define their own preferred digital signature schemes, including post-quantum signatures as recently standardized by NIST [5,11,26]. In all known schemes, post-quantum signatures or public keys are substantially larger than their classical counterparts. Since post-quantum signatures are generally expensive to verify on-chain, they are prime candidates for the naysayer proof paradigm.

CRYSTALS-Dilithium [11]. The verifier of this scheme checks that the following holds for signature $\sigma = (\mathbf{z}, c)$, public key $pk = (\mathbf{A}, \mathbf{t})$, and message M:

$$\forall i : \|z_i\|_\infty \leq C \wedge \mathbf{A}\mathbf{z} - c\mathbf{t} = \mathbf{w} \wedge c = H(M||\mathbf{w}), \tag{3}$$

where C is a constant, $\mathbf{A} \in R_q^{k \times l}$, and $\mathbf{z}, \mathbf{t}, \mathbf{w} \in R_q^k$ for the polynomial ring $R_q := \mathbb{Z}_q[x]/(X^{256} + 1)$. Notice that the checks in Equation (3) are efficiently naysayable. In fact, the naysayer prover must show that the following holds:

$$\exists i : \|z_i\|_\infty > C \vee \mathbf{A}\mathbf{z} - c\mathbf{t} \neq \mathbf{w} \vee c \neq H(M||\mathbf{w}). \tag{4}$$

If the first check fails, then the naysayer prover shows an index i for which the infinity norm of one of the polynomials in $\mathbf{z}$ is large. If the second check fails, then the naysayer prover can point to the ith row of the vector $\mathbf{w}$, where matrix-vector multiplication fails and verify only that row. Finally, if the last check fails, then the naysayer verifier just needs to recompute a single hash evaluation.

SPHINCS+ [5]. The signature verifier in SPHINCS+ checks several Merkle authentication proofs, requiring hundreds of hash evaluations. A constant-size and -time naysayer proof can be easily devised akin to the naysayer proof described in Section 4.1. The naysayer prover simply points to the hash evaluation in one of the Merkle-trees where the signature verification fails.

4.3 Verifiable shuffles

Verifiable shuffles are applied in many (blockchain) applications such as single secret leader election algorithms [6], mix-nets [8], cryptocurrency mixers [29], and e-voting [1]. The state-of-the-art proof system for proving the correctness of a shuffle is due to Bayer and Groth [2]. 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 succinct naysayer proof if the naysayer knows at least one of the shuffled elements. Let us consider the simplest case of shuffling public keys. We want to prove membership in the following NP language:

$$\mathcal{R}_{perm} := \{(g^{w_i}, g^{r \cdot w_{\sigma(i)}})_{i=1}^{n}, g^r; \sigma, r | \ \forall i : w_i, r \in_R \mathbb{F}_p, g \in \mathbb{G}, \sigma \in_R \mathsf{Perm}(n)\}, \tag{5}$$

where $\mathsf{Perm}(n)$ is the set of all permutations $f : [n] \to [n]$. Suppose the naysayer knows that for $j \in [n]$, the prover did not correctly include $g^{r \cdot w_j}$ in the shuffle. The naysayer can prove this by showing that $(g, g^{w_j}, g^r, g^{r \cdot w_j}) \in \mathcal{R}_{DH} \wedge g^{r \cdot w_j} \notin (\cdot, g^{r \cdot w_{\sigma(i)}})_{i=1}^{n}$, where $\mathcal{R}_{DH}$ is the language of Diffie-Hellman tuples. One can show that a tuple is a Diffie-Hellman tuple with a proof of knowledge of discrete logarithm equality [9]. However, the naysayer must know the discrete logarithm w_j to produce such a proof. Unlike our previous examples, which were publicly naysayable, this is a privately naysayable proof since the naysayer algorithm takes auxiliary input w_j. With the right data structure for the permuted list (e.g., a hash table), both of the above conditions can be checked in constant-time with a constant-size naysayer proof, resulting in exponential savings compared to directly verifying the original Bayer-Groth shuffle proof.

We evaluate the asymptotic cost savings for the verifiers in the four examples discussed in Section 4. Note that naysayer proofs allow an exponential speedup for the verifier for verifiable shuffles and a logarithmic speedup for the FRI polynomial commitment opening proof verifier, see Table 1. For CRYSTALS-Dilithium, we can only claim weakly efficient naysayer proofs, as there is no asymptotic gap in the complexity in certain branches of the signature verification circuit and the naysayer prover algorithm, cf. Equations (3) and (4).

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 only needs to compute $\mathsf{VerifyNay}$ 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

	FRI Opening	**CRYSTALS-D.**	**SPHINCS+**	**Shuffle proof**
π storage	$\mathcal{O}(\lambda \log^2(d))\mathbb{H}$	$\mathcal{O}(\lambda)\mathbb{F}$	$\mathcal{O}(\lambda)\mathbb{F}$	$\mathcal{O}(\sqrt{n})\mathbb{G}$
$\mathsf{Verify}(\pi)$ compute	$\mathcal{O}(\lambda \log^2(d))\mathbb{H}$	$\mathcal{O}(\lambda)\mathbb{F} + 1\mathbb{H}$	$\mathcal{O}(\lambda)\mathbb{H}$	$\mathcal{O}(n)\mathbb{G}$
π_{nay} storage	$1\mathbb{F}$	$1\mathbb{F} \vee 1\mathbb{F} \vee 1\mathbb{F}$	$1\mathbb{F}$	$2\mathbb{G} + 1\mathbb{F}$
$\mathsf{NVerify}(\pi_{nay})$ compute	$1\mathbb{H}$	$\mathcal{O}(\lambda)\mathbb{F} \vee \mathcal{O}(\lambda)\mathbb{F} \vee 1\mathbb{H}$	$1\mathbb{H}$	$4\mathbb{G}$

Table 1. Cost savings of the naysayer paradigm for the example applications in Section 4. In FRI, let $deg(p(x)) = d$. For the Bayer-Groth shuffle argument [2], we consider n shuffled public keys (or ciphertexts). $\mathbb{F}, \mathbb{G}$ denotes field/group elements or field/group operations, respectively. $\mathbb{H}$ denotes hashing operations.

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 [7,20] or polynomial [18], 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.

This suggests an optimization for naysayer proofs in a blockchain context: the prover posts only a binding commitment $H(\pi)$, which the contract stores in S_{per}, while the actual proof π 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 $H(\pi)$ stored in S_{per}. Note that in some naysayer proof systems which don't require reading all of π, 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 for naysayer proofs.

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 Mahimna Kelkar, Joachim Neu, Valeria Nikolaenko, Ron Rothblum, Ertem Nusret Tas, and Justin Thaler for insightful discussions. This work was supported by a16z 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, a16z, or any other supporting organization.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security 2008. pp. 335–348. USENIX Association (Jul / Aug 2008)
2. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4˙17
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://eprint.iacr.org/2018/046
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: On the concrete efficiency of probabilistically-checkable proofs. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 585–594. ACM Press (Jun 2013). https://doi.org/10.1145/2488608.2488681
5. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS$^+$ signature framework. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2129–2146. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363229
6. Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. In: Advances in Financial Technologies (2020)
7. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7˙5
8. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **24**(2) (1981)
9. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4˙7
10. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC (1971)
11. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
12. Ethereum: Account Abstraction (2023), ethereum.org/en/roadmap/account-abstraction

13. Ethereum: Optimistic rollups (2023), ethereum.org/en/developers/docs/scaling/optimistic-rollups/
14. Feige, U., Kilian, J.: Making games short (extended abstract). In: 29th ACM STOC. pp. 506–516. ACM Press (May 1997). https://doi.org/10.1145/258533.258644
15. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
16. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7˙25
17. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5˙11
18. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8˙11
19. Labs, O.: Inside Arbitrum Nitro (2023), docs.arbitrum.io/inside-arbitrum-nitro/
20. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (Aug 1988). https://doi.org/10.1007/3-540-48184-2˙32
21. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review (2008)
22. Optimism: Rollup Protocol (2023), community.optimism.io/docs/protocol/2-rollup-protocol
23. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) ITCS 2019. vol. 124, pp. 60:1–60:15. LIPIcs (Jan 2019). https://doi.org/10.4230/LIPIcs.ITCS.2019.60
24. Poon, J., Buterin, V.: Plasma: Scalable autonomous smart contracts (2017)
25. Poon, J., Dryja, T.: The Bitcoin lightning network: Scalable off-chain instant payments. lightning.network/lightning-network-paper.pdf (2016)
26. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
27. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: ETHDKG: Distributed key generation with Ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985 (2019), https://eprint.iacr.org/2019/985
28. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0˙22
29. Seres, I.A., Nagy, D.A., Buckland, C., Burcsi, P.: MixEth: efficient, trustless coin mixing service for Ethereum. Cryptology ePrint Archive, Report 2019/341 (2019), https://eprint.iacr.org/2019/341
30. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains. arXiv preprint arXiv:1908.04756 (2019)
31. Thaler, J.: Proofs, arguments, and zero-knowledge (July 2023), https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

32. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper (2014)