Feta: Efficient Threshold Designated-Verifier Zero-Knowledge Proofs

Carsten Baum[1], Robin Jadoul[2], Emmanuela Orsini[2], Peter Scholl[1], and Nigel P. Smart[2]

[1] Dept. Computer Science, Aarhus University, Aarhus, Denmark.
[2] imec-COSIC, KU Leuven, Leuven, Belgium. `cbaum@cs.au.dk`, `robin.jadoul@esat.kuleuven.be`, `emmanuela.orsini@kuleuven.be`, `peter.scholl@cs.au.dk`, `nigel.smart@kuleuven.be`,

Abstract. Zero-Knowledge protocols have increasingly become both popular and practical in recent years due to their applicability in many areas such as blockchain systems. Unfortunately, public verifiability and small proof sizes of zero-knowledge protocols currently come at the price of strong assumptions, large prover time, or both, when considering statements with millions of gates. In this regime, the most prover-efficient protocols are in the designated verifier setting, where proofs are only valid to a single party that must keep a secret state.

In this work, we bridge this gap between designated-verifier proofs and public verifiability by *distributing the verifier*. Here, a set of verifiers can then verify a proof and, if a given threshold t of the n verifiers is honest and trusted, can act as guarantors for the validity of a statement. We achieve this while keeping the concrete efficiency of current designated-verifier proofs, and present constructions that have small concrete computation and communication cost. We present practical protocols in the setting of threshold verifiers with $t < n/4$ and $t < n/3$, for which we give performance figures, showcasing the efficiency of our approach.

Table of Contents

1 Introduction

A zero-knowledge proof of knowledge (ZKPoK) is an interactive protocol which allows a prover to convince a verifier, given a statement x, that the prover knows a witness w such that the pair (x, w) lies in some NP language $\mathcal{L}$. This is done in such a way that the verifier learns nothing but the validity of the statement, i.e. they learn nothing about the witness w, only that the prover knows the witness. ZKPoKs have a wide range of applications, especially in the burgeoning area of blockchain [HBHW16], but also as building blocks of highly efficient signature schemes [CDG$^+$17] or to increase the security level of existing cryptographic protocols from passive to active security in a black-box manner [GMW87].

There are various parameters that influence which ZKPoK scheme is suitable for a certain application. For example, when using ZKPoKs for blockchains one needs proofs that are *publicly verifiable* and *non-interactive*; namely the proof is sent in a single message from the prover such that any verifier can verify it. Another common requirement is that they are *succint*; namely that the proof has size and verification time that is sublinear in the size of the statement.

Therefore, most ZKPoKs such as SNARKs [BCG$^+$13] and STARKs [BBHR19] that are considered for practical applications within blockchains for instance, are mainly optimized for small proof size and verification time (and are also publicly verifiable and non-interactive). Their drawback is that prover running time can be prohibitive for large statements, i.e. statements expressed by arithmetic circuits with billions of gates. This is because the prover runtime for all current practical succinct schemes has an inherent $polylog(|x|)$ overhead over the optimal $O(|x|)$ proof time and because prover memory access is not local[3], which leads to inherent slowdowns for increasing $|x|$.

MPC-in-the-Head ZKPoKs such as KKW [KKW18] or Limbo [dSGOT21] have a proof size that is at least linear in $|x|$, with the unique exception of Ligero [AHIV17] which achieves sub-linear proof for large enough statements. In addition, they usually use a "light" inner proof (which is a passively-secure MPC scheme) that requires $O(|x|)$ computation, but must be repeated $s/\log(s)$ times to achieve negligible soundness error where s is the security parameter.

Alternative ZKPoKs for large statements, which also have a practically efficient prover due to small concrete constants, are either based on garbled circuits ([JKO13] and follow-ups) or vOLE-commitments [WYKW21, YSWW21, BMRS21]). All of these prover-efficient schemes have the disadvantage that they require the verifier to keep a secret state, i.e. they are designated-verifier ZKPoKs. This means that the proof can only be verified by a single party, who must be identified before the proof is produced. This makes the application in blockchains, where a proof may need to be verified by a set of validator nodes, impossible.

One can mitigate the problem of a designated verifier by distributing the verification among a larger set of parties. Here, each such verifier comes from a pre-defined, possibly large set, leading to a form of distributed designated verifier proof system. Now, if a majority of these verifiers is trusted, the statement of the prover can be accepted as validated by a majority of third parties.

Distributing Verification. This distribution of verification has an impact on the question of what a proof actually is, and also changes how protocols for such a setting can be designed.

- If the verifier is distributed, an adversary may corrupt multiple verifiers, in addition to the prover, in order to convince honest verifiers of the validity of a false statement. This means that soundness must be redefined to take this into consideration.
- When a proof is rejected, this might happen either if a prover does not have a proof or if it is honest, but verifiers may prevent successful verification of a proof. Hence, honest verifiers may want to distinguish these cases in order to not blame an honest prover or verifier as corrupt. So in the case of dishonest behaviour a security definition may require that honest verifiers do not just abort, but they also identify one (or more) of the cheating parties. This enables a form of cheater elimination.
- The distributed nature of the verifier may allow to obtain more efficient protocols: while in standard zero-knowledge the verifier must always be considered as fully corrupted, we may now be ok with only maintaining zero-knowledge if a strict subset of the verifiers does not collude.

1.1 Our Contribution

In this work, we are the first to formalize the notion of Distributed Verifier ZKPoKs (DV-ZKPoKs). We provide multiple constructions of such protocols, all with cheater identification, that are secure against different thresholds of corrupted verifiers[4].

[3] There are theoretical works that achieve linear prover time such as e.g. [LSTW21], but to the best of our knowledge they are not concretely efficient.

[4] In our construction, the single (cheesy) verifier of the Mac-and-Cheese protocol [BMRS21] has been crumbled into a large set of possibly smaller verifiers. Thus, our protocol name *Feta*.

New definitions. We first present a formal definition of what it means for a Distributed verifier ZKPoK to be secure. We redefine the three standard properties of ZKPoKs to be applicable to the threshold setting:

Distributed Correctness: If the prover has a witness, then the honest parties either accept the proof or identify the same corrupted verifiers that interfered with the proof.

Distributed Soundness: If the prover does not have a witness then honest verifiers only accept with negligible probability, given not too many other verifiers are corrupted. In addition the honest verifiers either agree that the prover does not have a witness, or will identify a set of corrupted verifiers.

Distributed Zero-Knowledge: The corrupted verifiers learn no new information beyond the fact that the statement is true.

Our definition will allow different adversarial structures for all of these properties. This means that our definition also encapsulates protocols where e.g. soundness breaks down if just one verifier is corrupted, but which are zero-knowledge even if all verifiers are corrupted.

There are a number of "naive" protocols which enable such distributed verifier zero-knowledge proofs using existing techniques. We will describe some of these protocols, showing the applicability of our framework.

New protocols. We then present two efficient DV-ZKPoK protocols together with necessary preprocessing protocols. These protocols are optimized for $t < n/4$ and $t < n/3$ corruptions, respectively, where n is the number of verifiers and t is the number of corrupted verifiers. Our protocols are plausibly post-quantum secure, and require as setup assumptions a PKI as well as a broadcast channel. The latter can easily be implemented if $t < n/3$ information theoretically.

Implementations. We have implemented our protocols in C++. Our protocol for the case of $t < n/4$ is very efficient both in terms of prover and verifier time. For example, the combined pre-processing and prover time for proving knowledge of the pre-image of a single SHA-256 evaluation with $n = 5$ verifiers is under 9 18 milliseconds, with a proof time of under 6.5 milliseconds. The verification time is under 10 milliseconds. This is with a single threaded implementation of our protocols.

Our run times are all significantly smaller than the single instance publicly-verifiable proofs of such a pre-image using a system such as Ligero [AHIV17]. Using machines less powerful than the ones we used in our experiments, [AHIV17] give prover and verification times for a single pre-image of a SHA-256 evaluation of over 100 milliseconds. Our proof size, excluding pre-processing, is also significantly smaller (8 KBytes vs 100's of KBytes for Ligero). Note, Ligero provides a publicly verifiable proof as opposed to our distributed designated verifier proofs.

The Limbo system [dSGOT21], which again provides publicly verifiable proofs, reports single threaded prover and verifier times for the same circuit of 50 milliseconds, using machines comparable to the ones in our experiments. With their proof sizes being 42 KBytes.

The Mac-n-Cheese [WYKW21] and Quicksilver protocols [BMRS21], which provide designated verifier proofs using a single threaded implementation can achieve around 7 million AND gates per second in terms of prover/verification time. Translating this to the 22.573 AND gate SHA-256 circuit would equate to a prover/verification time of 3 milliseconds.

Thus, we see our prover/verification time of 6.5/10 milliseconds, for the SHA-256 circuit in the distributed verifier case, provides a compromise between slower publicly verifiable proofs and faster designated verifier proofs.

The protocol for the case of $t < n/3$ is less efficient, but still provides a highly efficient methodology for performing distributed verifier zero-knowledge proofs. The verification time is comparable to that of Limbo, with a prover time roughly three times more expensive.

Hence, we see that our notion of distributed designated verifier proofs can enable more efficient practical zero-knowledge proofs when compared to publicly verifiable proofs.

1.2 Techniques

On a high level, our protocols can be described using the following four-step paradigm:

1. The verifiers create consistent commitments to random values r_i such that only the prover can open these later. Here, if t or less verifiers are corrupted, then they cannot reconstruct the committed values themselves.
2. The verifiers and the prover check together that the commitments to the random values are indeed consistent among all verifiers, and that the prover knows the openings. If not, then cheaters are identified. If they are consistent, then the preprocessing of the DV-ZKPoK is considered as finished.
3. In the online phase, the prover uses the r_i to commit to w as well as auxiliary information necessary to show that $(x, w) \in R$. This commitment can ideally be done by sending one message via a broadcast channel.
4. Upon the prover having finished committing, the verifiers perform a proof verification step. Here we aim for a "cheap" proof verification that only requires the verifiers to communicate in $O(1)$ rounds, with a message complexity that is sublinear in $|x|$ or $|w|$ as well.

To achieve this, our "preprocessing" phase lets the verifiers create many random Shamir secret sharings as commitments, where the prover only learns the secret being shared. Given the linearity of this secret sharing, consistency can easily be established using a linear test. This test only requires communication that scales in the number of parties but not $|x|$ or $|w|$. Moreover, we show that cheater identification can be achieved by additionally signing certain messages in the preprocessing protocol.

In our online phase, our protocols let the prover commit both to w as well as the intermediate wire values for a circuit C that evaluates to 0 iff w is a valid witness for the statement x. The verifiers re-evaluate C based on the committed w using the homomorphic properties of the commitment/secret sharing and check if the intermediate wire values are consistent with w and that the output of $C(w)$ is 0. This only requires a depth-1 circuit to be evaluated by the verifiers.

In the first protocol (for $t < n/4$) we make use of error-detecting properties of a Reed-Solomon code/Shamir sharing. The linear gates are free to evaluate as the Shamir sharing is linearly homomorphic, while the multiplication is performed by each verifier multiplying the input shares of a multiplication gate locally. The bound of $t < n/4$ comes from having to perform error detection on product codes (coming from degree $2 \cdot t$ polynomials stemming from the share multiplication), which is necessary to detect cheating during the multiplication protocol by a verifier.

Our second protocol (for $t < n/3$) is slightly more complex and avoids the verifiers having to multiply shares altogether. Instead we let the prover commit to slightly more data and use a checking procedure for multiplications that is based on the Schwarz-Zippel Lemma. This means that multiplication checks only require linear operations.

1.3 Related Work

Thresholdizing in Zero-Knowledge proofs has appeared in previous work, although different from how we consider it.

A related notion is the concept of distributed zero-knowedge notion from [BBC$^+$19], which looks at the case where the *statement* x is unknown to any given verifier, and is instead secret shared. The protocols in that work only support a limited class of languages, and do not consider identifiable abort, so are vulnerable to denial-of-service attacks from a malicious verifier. Our notion can be seen as orthogonal to Multi-Prover Interactive Proofs [BGKW88], where multiple provers act independently to convince a verifier. Our notion is also complementary to the setting considered in [WZC$^+$18] where the witness w is shared amongst a set of provers. Instead, we only have one prover and w is shared among the verifiers.

Conceptually, our setting bears resemblance to the one considered in the MPC-in-the-head paradigm [IKOS07] where the proof is verified by a set of simulated verifiers. In comparison to [IKOS07] our setting requires that the adversarial prover can only cooperate with a small set of corrupt verifiers, such that that some of the verifiers have a secret state unbeknownst to the prover.

2 Preliminaries

2.1 Shamir Sharing

Our protocols are built on top of Shamir's secret-sharing scheme [Sha79]. We briefly recap on it here in order to fix the notation we will use in the rest of the paper.

A secret s, in a finite field $\mathbb{F}$, is shared amongst n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ by the sharing party defining a random degree t polynomial $f_s(X)$ whose constant term is the value s. Assuming $n > |\mathbb{F}|$ and that the integers $\{1, \ldots, n\}$ are mapped to distinct non-zero values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$, each party P_i is given the share $s^{(i)} = f_s(\alpha_i) \in \mathbb{F}$. We denote such a sharing by $\langle s \rangle_t$.

Note that this secret sharing scheme is linear, namely given $\beta, \delta, \gamma \in \mathbb{F}$ and two sharings $\langle x \rangle_t$ and $\langle y \rangle_t$, both of degree t, parties can locally produce the sharing $\langle z \rangle_t$, where $z = \beta \cdot x + \delta \cdot y + \gamma$, by computing

$$z^{(i)} = \beta \cdot x^{(i)} + \delta \cdot y^{(i)} + \gamma.$$

Also note that one can linearly combine sharings of different degrees to produce a sharing of the maximal degree, i.e. given $\langle x \rangle_{t_1}$ and $\langle y \rangle_{t_2}$ then one can locally produce $\langle x{+}y \rangle_t$, where $t = \max(t_1, t_2)$, which we shall write as $\langle x \rangle_{t_1} + \langle y \rangle_{t_2}$.

Reconstruction of a secret s, shared via $\langle s \rangle_t$, requires $t + 1$ correct share values from different parties. It is well known that Shamir's secret sharing scheme defined as above is equivalent to a Reed-Solomon code $[n, t + 1, n - t]$ over $\mathbb{F}$, where the shares $(f_s(\alpha_1), \ldots, f_s(\alpha_n))$ are viewed as a codeword. In particular, when the number of dishonest parties is bounded by d and $n > t + 2 \cdot d$, the parties can robustly reconstruct a shared value $\langle s \rangle_t$, so that any party who lies about their sharings will be detected. In one of our protocols we will use the fact that, if $n > 4 \cdot t$ and $d < t$ we can robustly reconstruct a value for a sharing of degree $2 \cdot t$.

Assuming $n > t + 2 \cdot d$, we denote by $\mathsf{RobustReconstruct}(\langle s \rangle_t, d)$ the reconstruction algorithm associated with Shamir's scheme which outputs a pair (s, flag), where either $\mathsf{flag} = (\mathsf{correct}, \emptyset)$, indicating that all the shares are consistent with a degree t sharing, or $\mathsf{flag} = (\mathsf{incorrect}, \mathcal{D})$ where $\mathcal{D}$ indicates the parties who input an inconsistent shares.

2.2 Digital Signatures

Our basic protocols will make use of digital signatures, for which we use the following two standard definitions.

Definition 1. *A digital signature scheme for message space $\mathcal{M}$ is given by a triple of polynomial time algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$.

- $\mathsf{KeyGen}(1^\lambda)$**:** *On input a security parameter λ this randomized algorithm outputs a public/private key pair $(\mathfrak{pk}, \mathfrak{sk})$.*
- $\mathsf{Sign}(\mathfrak{sk}, m)$**:** *On input of private key $\mathfrak{sk}$ and a message $m \in \mathcal{M}$, this (potentially) randomized algorithm outputs a digital signature σ.*
- $\mathsf{Verify}(\mathfrak{pk}, \sigma, m)$**:** *On input of a public key $\mathfrak{pk}$, a message m and a purported signature σ, this algorithm outputs either* true *(meaning accept the signature) or* false *(meaning reject the signature).*

A digital signature scheme is said to be correct if for each $m \leftarrow \mathcal{M}$ and $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$,

$$\mathsf{Verify}(\mathfrak{pk}, \mathsf{Sign}(\mathfrak{sk}, m), m) = \mathsf{true}.$$

A digital signature scheme is said to be UF-CMA secure if the probability of any adversary $\mathcal{A}$ winning the following game is negligible in λ

1. $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathfrak{sk}, \cdot)}(\mathfrak{pk})$.
3. Output 'win' if and only if $\mathsf{Verify}(\mathfrak{pk}, \sigma^*, m^*) = \mathsf{true}$ and m^* was not queried to $\mathcal{A}$'s signing oracle.

2.3 Zero-knowledge Proofs

A standard zero-knowledge proof takes a statement x and a witness w from some NP relation $\mathcal{R}$. The prover $\mathcal{P}$ holds the pair $(x, w) \in \mathcal{R}$, whilst the verifier only has x. The goal of a zero-knowledge proof (of knowledge) is to convince the verifier that x is in the language $\mathcal{L}_\mathcal{R}$ of statements that have a witness in $\mathcal{R}$. This is done by asserting that the prover holds w such that $(x, w) \in \mathcal{R}$, while no information about w (bar the fact that the prover knows it) is revealed to the verifier. Informally, a zero-knowledge proof has three security properties:

Correctness: If $(x, w) \in \mathcal{R}$ then $\mathcal{V}$ always accepts.
Soundness: If $\mathcal{P}$ does not have w then $\mathcal{V}$ only accepts with negligible probability.
Zero-Knowledge: There exists a simulator $\mathcal{S}$ that on input x can create transcripts of protocol instances between $\mathcal{P}$ and $\mathcal{V}$ that make $\mathcal{V}$ accept.

In the designated verifier setting, the soundness only holds for a verifier that has a secret state.

2.4 Schwarz-Zippel Lemma

One of our protocols will make use of the Schwarz-Zippel lemma for univariate polynomials, which we state here.

Lemma 1 (Schwartz-Zippel Lemma). *Let $F \in \mathbb{F}[X]$ denote a non-zero polynomial of degree d over a field $\mathbb{F}$. Let S denote a finite subset of elements of $\mathbb{F}$. If one selects $r \in S$ uniformly at random then*

$$\Pr[\ F(r) = 0\ ] \le \frac{d}{|S|}.$$

2.5 Coin Flipping

We will utilize at various points the ideal functionality $\mathcal{F}_{\text{Rand}}(\mathcal{P}, M, \mathbb{F})$, described in Fig. 1. This functionality allows a set of parties $\mathcal{P}$ to sample M uniformly random values from a finite field $\mathbb{F}$ such that each party learns these. It does this in a manner which has identifiable abort, in the case that the adversary aborts the execution of the protocol. The implementation of this functionality is standard: The parties agree on a shared single seed using a non-interactive commitment via broadcast, then open via broadcast, and then the seed is expanded into the desired number of random values from $\mathbb{F}$ using a PRG.

The Ideal $\mathcal{F}_{\text{Rand}}(\mathcal{P}, M, \mathbb{F})$ Functionality

On input $(\mathsf{Rand}, \mathsf{cnt})$ from all parties in $\mathcal{P}$, if the counter value is the same for all parties and has not been used before:

1. Sample $r_i \leftarrow \mathbb{F}$ for $i \in [M]$.
2. The values r_i are sent to the adversary, and the functionality waits for its input.
3. If the input is $\mathsf{Deliver}$ then the values r_i are sent to all parties. Otherwise the adversary will return a non-trivial subset C_A of the dishonest parties. The value (Abort, C_A) is returned to all parties.

Fig. 1. Functionality $\mathcal{F}_{\text{Rand}}(\mathcal{P}, M)$

3 Distributed Verifier Zero-Knowledge Proofs

Our definition of *Distributed Verifier Zero-Knowledge Proofs* (DV-ZKPoKs) aims to generalize the notion of a *Designated Verifier Zero-Knowledge Proof* to the threshold setting. Namely, we will have a set of designated verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ who jointly verify the correctness of the proof using an interactive protocol.

3.1 Zero-Knowledge in the Threshold Setting

As mentioned in Section 1 in a distributed verifier setting there might exist multiple verifiers $\mathcal{V}_i$, some of whom may collaborate with a potentially corrupt prover $\mathcal{P}$. For a DV-ZKPoK we therefore get the following intuitive properties.

Distributed Correctness: If $(x, w) \in \mathcal{R}$ then either all honest verifiers $\mathcal{V}$ always accept or all honest verifiers agree on a set of cheating verifiers C_A.

Distributed Soundness: If $\mathcal{P}$ does not have w then honest verifiers only accept with negligible probability.

Distributed Zero-Knowledge: There exists a simulator $\mathcal{S}$ that on input x can create transcripts of protocol instances between $\mathcal{P}$ and verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ that make verifiers accept.

Let $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$ denote the set of verifiers. An *access structure* Γ on $\mathcal{V}$ is a monotonically increasing subset of $2^{\mathcal{V}}$, i.e., if $S \in \Gamma$ then we have $T \in \Gamma$ for all T such that $S \subseteq T \subseteq \mathcal{V}$. The *adversary structure* Δ associated with Γ is the set of all sets $\mathcal{V} \setminus S$ for $S \in \Gamma$.

When dealing with a potentially dishonest prover and a subset of potentially dishonest verifiers, we can consider three different access structures related to the three different properties of ZK proofs. We let the relevant access structures, for the potentially dishonest verifiers, be denoted by Γ_C (for Correctness), Γ_S (for Soundness) and Γ_Z (for Zero-Knowledge). With their different associated adversary structures being Δ_C, Δ_S and Δ_Z. We allow different access structures to provide better flexibility in applications, as well as more flexibility in designing protocols. To aid the reader one could initially think of the threshold case of $\Gamma_C = \Gamma_S = \Gamma_Z$ being all subsets of size greater than $n - t$, and $\Delta_C = \Delta_S = \Delta_Z$ being all subsets of the verifiers of size less than or equal to t.

We let $\mathcal{V}_{\mathcal{D}}$ denote the precise set of dishonest verifiers in a given protocol instance. We desire that at the end of the protocol, the verifiers either output Abort, $\mathsf{Success}$ or Fail. Here, $\mathsf{Success}$ or Fail imply that the proof was correct or not, respectively, while Abort means that some verifiers or the prover may have aborted. In all cases each honest party P will obtain a non-empty list of parties who aborted.

Distributed Correctness. We first discuss correctness; as usual this assumes an honest prover. In the case of $\mathcal{V}_{\mathcal{D}} \not\in \Delta_C$ then the adversary has enough power to break correctness. In this case some honest verifiers will abort, some will accept and some will fail - no common guarantees can be made. Note in the case when $\mathcal{V}_{\mathcal{D}} \not\in \Delta_C$, the set C that each honest verifier identifies as corrupt parties in the case of abort, can be different for each of them, and they may even identify honest parties as corrupted. In the case of failure or success the honest verifiers may in addition identify cheating verifiers. This is captured by the procedure $\mathsf{Breakdown}()$ in our ideal functionality $\mathcal{F}_{\mathrm{DV-ZK}}$, which can be found in Fig. 2.

However, when $\mathcal{V}_{\mathcal{D}} \in \Delta_C$ then the parties obtain consensus of output: either all honest verifiers output $\mathsf{Success}$ or they all output Abort. In the latter case, the verifiers identify a set $C_A \neq \emptyset$ of dishonest verifiers which is the same for each honest verifier. Consensus of output when $\mathcal{V}_{\mathcal{D}} \in \Delta_C$ is needed to avoid denial-of-service attacks where a single dishonest verifier can make the honest verifiers reject a valid proof. This is captured by the procedure $\mathsf{CompleteWithAbort}()$ in our ideal functionality $\mathcal{F}_{\mathrm{DV-ZK}}$.

Note that cheater identification is not necessary in the case of honest majority access structures Γ_C. This is because a simple majority vote will result in the honest verifiers accepting the proof (assuming consensus on accept). In the case of dishonest majority the ability for the honest parties to identify a single dishonest party (with consensus) will act as a deterrent to verifiers to act dishonestly. Thus even in the case of acceptance we allow the identification of dishonest verifiers so as to allow our functionality to capture the dishonest majority case.

Distributed Soundness. Soundness considers the case of a dishonest prover. We require that if $\mathcal{V}_{\mathcal{D}} \not\in \Delta_S$ then the adversary can get the honest verifiers to output anything it wants. Which is again captured by the procedure $\mathsf{Breakdown}()$ in Fig. 2.

As we require the prover to input a witness w, if $\mathcal{V}_{\mathcal{D}} \in \Delta_S$ and if $(x, w) \in \mathcal{R}$ then the *worst* $\mathcal{P}$ can do is get some honest verifiers to abort and identify a cheating party. This is again captured by the procedure $\mathsf{CompleteWithAbort}()$ in Fig. 2. On the other hand, if $(x, w) \not\in \mathcal{R}$ then the *best* $\mathcal{P}$ can achieve is to get some honest verifiers to abort and identify a cheating party (which could include the prover). Again, this is captured by the procedure $\mathsf{FailWithAbort}()$ in Fig. 2.

Functionality $\mathcal{F}_{\mathrm{DV-ZK}}$

This functionality communicates with $n+1$ parties $\mathcal{P}, \mathcal{V}_1, \dots, \mathcal{V}_n$ as well as the ideal adversary $\mathcal{S}$. We call $\mathcal{P}$ the prover and $\mathcal{V} = \{\mathcal{V}_1, \dots, \mathcal{V}_n\}$ the verifiers. For simplicity, we write $\mathcal{W} = \mathcal{V} \cup \{\mathcal{P}\}$. The functionality is instantiated with descriptions of three access structures $\Gamma_C, \Gamma_S, \Gamma_Z \subseteq 2^{\mathcal{V}}$, and their associated adversary structures Δ_C, Δ_S and Δ_Z. The adversary structures denote which parties $\mathcal{S}$ can corrupt without leading to a loss of correctness, soundness or zero-knowledge. Let `init` be a flag that is initially $\bot$.

Corrupt: Before any other command, $\mathcal{S}$ sends $(\mathsf{Corrupt}, \mathcal{D})$ where $\mathcal{D} \subseteq \mathcal{W}$. Let $\mathcal{H} = \mathcal{W} \setminus \mathcal{D}$. If $\mathcal{P} \in \mathcal{D}$ then we call the prover "corrupted", otherwise "honest". We call $\mathcal{V}_{\mathcal{D}} = \mathcal{V} \cap \mathcal{D}$ the corrupted verifiers and $\mathcal{V}_{\mathcal{H}} = \mathcal{V} \setminus \mathcal{V}_{\mathcal{D}}$ the honest verifiers.

Init: On input (Init) by all parties in $\mathcal{H}$:

1. Send $(\mathsf{Init?})$ to $\mathcal{S}$. If $\mathcal{S}$ responds with (ok) then send (InitOK) to all parties in $\mathcal{H}$ and set `init` $\leftarrow \top$. Otherwise send (Abort) to all parties in $\mathcal{H}$.

ProveHonest: On input (Prove, x, w) by $\mathcal{P} \in \mathcal{H}$ as well as (Prove, x) by all parties in $\mathcal{V}_{\mathcal{H}}$, if `init` $= \top$ and if $(x, w) \in R_L$:

1. If $\mathcal{V}_{\mathcal{D}} \notin \Delta_Z$ then send $(\mathsf{Prove?}, x, w)$ to $\mathcal{S}$, otherwise send $(\mathsf{Prove?}, x)$.
   - If $\mathcal{V}_{\mathcal{D}} \notin \Delta_C$ then run $\mathsf{Breakdown}()$.
   - If $\mathcal{V}_{\mathcal{D}} \in \Delta_C$ then run $\mathsf{CompleteWithAbort}()$.

ProveDishonest: On input (Prove, x, w) by $\mathcal{S}$ if $\mathcal{P} \in \mathcal{D}$ as well as (Prove, x) by all parties in $\mathcal{V}_{\mathcal{H}}$ and if `init` $= \top$:

- If $\mathcal{V}_{\mathcal{D}} \notin \Delta_S$ or $\mathcal{V}_{\mathcal{D}} \notin \Delta_C$ then run $\mathsf{Breakdown}()$.
- If $\mathcal{V}_{\mathcal{D}} \in \Delta_S$, $\mathcal{V}_{\mathcal{D}} \in \Delta_C$ and $(x, w) \in R_L$ then run $\mathsf{CompleteWithAbort}()$.
- If $\mathcal{V}_{\mathcal{D}} \in \Delta_S$, $\mathcal{V}_{\mathcal{D}} \in \Delta_C$ and $(x, w) \notin R_L$ then run $\mathsf{FailWithAbort}()$.

Method $\mathsf{Breakdown}()$:

1. Wait for a message $(\mathsf{Abort}, A, F, S, C)$ from $\mathcal{S}$ where A, F, S are disjunct sets, $A \cup F \cup S = \mathcal{H}$, $C_A : \mathcal{H} \to 2^{\mathcal{W}}$.
2. Send $(\mathsf{Abort}, x, C_A(P))$ to each $P \in A$, $(\mathsf{Fail}, x, C_A(P))$ to each $P \in F$ and $(\mathsf{Success}, x, C_A(P))$ to each $P \in S$.

Method $\mathsf{CompleteWithAbort}()$:

1. Wait for a message (Abort, b, C_A) from $\mathcal{S}$ where $C_A \subseteq \mathcal{V}_{\mathcal{D}}$, $b \in \{0,1\}$ and $C_A \neq \emptyset$ if $b = 0$.
2. If $b = 0$ then send (Abort, x, C_A) to each $P \in \mathcal{H}$, otherwise send $(\mathsf{Success}, x, C_A)$ to each $P \in \mathcal{H}$.

Method $\mathsf{FailWithAbort}()$:

1. Wait for a message (Abort, b, C_A) from $\mathcal{S}$ where $C_A \subseteq \mathcal{V}_{\mathcal{D}}$, $b \in \{0,1\}$ and $C_A \neq \emptyset$ if $b = 0$.
2. If $b = 0$ then send (Abort, x, C_A) to each $P \in \mathcal{H}$, otherwise send (Fail, x, C_A) to each $P \in \mathcal{H}$.

Fig. 2. Functionality $\mathcal{F}_{\mathrm{DV-ZK}}$ for Distributed-Verifier ZK

Distributed Zero-Knowledge. Finally in the case of a honest prover, if $\mathcal{V}_{\mathcal{D}} \notin \Delta_Z$ then the adversary has enough power to break the zero-knowledge property and potentially learn information about w. But if $\mathcal{V}_{\mathcal{D}} \in \Delta_Z$ then the adversary cannot learn w.

It is straightforward to change $\mathcal{F}_{\mathrm{DV-ZK}}$ so that it only has unanimous abort. Another interesting strengthening is to not permit identifiable aborts if $\mathcal{V}_{\mathcal{D}} \in \Delta_C$. Since this setting seems to be not achievable if a majority of verifiers is corrupted for any interesting protocol[5], we have opted for a definition that is achievable in both the honest and dishonest-majority setting.

[5] It is achievable if the prover broadcasts a publicly verifiable proof to all verifiers. If the verifiers need to use a secret-shared state to validate the proof, then dishonest-majority completeness implies that $< n/2$ verifiers are sufficient to perform this validation and possibly reconstruct the secret state. But then, this implies that $< n/2$ corrupted verifiers can use their knowledge to aid a dishonest prover to break soundness.

3.2 Examples

We now explain the ideas behind our definition by presenting some naïve protocols that $\mathcal{F}_{\mathrm{DV-ZK}}$ captures, with different access structures Γ_C, Γ_S, and Γ_Z. In Table 1 we present a comparison of four "naïve" protocols, alongside our two more elaborate constructions, $\Pi_{4\mathrm{t}}$ and $\Pi_{3\mathrm{t}}$.

P0: Send a NIZK Assuming the existence of a functionality $\mathcal{F}_{\mathrm{NIZK}}$, as well as a broadcast channel, we can easily realize $\mathcal{F}_{\mathrm{DV-ZK}}$. There is no preprocessing (bar what is needed to set up the functionality $\mathcal{F}_{\mathrm{NIZK}}$) and the prover simply broadcasts the non-interactive proof. The verifiers then verify it using $\mathcal{F}_{\mathrm{NIZK}}$ and then come to consensus on the output. In the case of acceptance, any party who does not concur is determined to be an identified adversary. In that case $\Gamma_C = \Gamma_S = \Gamma_Z = \emptyset$, i.e. we can tolerate any set of adversaries possible. Without a broadcast channel, Γ_C and Γ_S instead follow from e.g. standard bounds on Byzantine agreement. The protocol can only be simulated if $\mathcal{F}_{\mathrm{NIZK}}$ is straight-line extractable.

P1: Secret-Share a Proof Suppose we have a single access structure Γ over the verifiers, we let $\langle\cdot\rangle$ denote an information theoretic secret sharing scheme which respects this access structure. A trivial protocol is to take a non-interactive two party ZKPoK, for the prover to generate a proof π and then simply generate a sharing $\langle\pi\rangle$ of that proof and distribute it to the verifiers. The verifiers then (simply) publish their received share.

In terms of correctness we require $\Gamma_C = \Gamma$ is $\mathcal{Q}_3$[6]. This follows as we require, in the presence of dishonest verifiers, that honest verifiers output either success with consensus, or output abort with consensus, and identify the cheater.

In terms of soundness we also require that $\Gamma_S = \Gamma$ is $\mathcal{Q}_3$, this follows as the proof π is already sound. Thus we require that for a (real or fake) proof that the verifiers come to a consensus and either identify a cheating verifier, or identify (in the case of a fake proof) that the prover has generated a fake proof.

In terms of zero-knowledge we have $\Gamma_Z = \emptyset$ since the initial proof π is zero-knowledge.

P2: Secret Share a Witness Instead of sharing the proof, the prover simply shares the witness according to some access structure Γ, and then the verifiers engage in an MPC protocol respecting Γ evaluating the circuit which verifies the witness. The zero-knowledge property is weaker than before, as we have $\Gamma_Z = \Gamma$. If the dishonest verifiers are not in the allowed adversary structure Δ then they can recover the witness and break the zero-knowledge property. The correctness, and the associated Γ_C, follow from the underlying MPC protocol (which needs to be a protocol which is either robust, or with identifiable abort). For soundness, and the associated Γ_S, we obtain $\Gamma_S = \Gamma_C$ by the correctness of the MPC protocol.

The advantage of this example, over P1 is that the prover has *almost no overhead* over secret-sharing the witness - it itself is not required to compute any kind of proof. In comparison to this generic protocol is highly likely to be significantly less efficient than our specialized protocols $\Pi_{4\mathrm{t}}$ and $\Pi_{3\mathrm{t}}$, which can be seen as variants of this protocol idea. Our protocols $\Pi_{4\mathrm{t}}$ and $\Pi_{3\mathrm{t}}$ perform this optimization by removing the expensive circuit evaluation needed in a generic MPC solution; this is done at the expense of the prover needing to provide more share values for the circuit evaluation and not just sharing a witness.

[6] A $\mathcal{Q}_3$ access structure can be simply thought of as one which admits robust opening, see [HM97]

P3: Joint MPC It may seem from the previous examples that we always have $\Gamma_C = \Gamma_S$ but this does not have to be the case. Consider the following construction, where we assume an MPC protocol run between the prover and the verifiers. The verifiers have no input, but the prover inputs the witness w. The common output (for the verifiers) is the evaluation of the checking circuit on the witness, or an identified cheater.

The proof is interactively performed between the prover and the verifiers by running the MPC protocol. Consider the case where Γ_C is a threshold structure on the n verifiers, with threshold t_C. In this case we can have that $t_C < (n+1)/3$ (because the prover acts honestly) and we can use an information theoretic robust protocol to ensure correctness. This also ensures that we have $t_Z < (n+1)/3$.

Now consider Γ_S with a threshold structure with threshold t_S. For the same protocol and soundness we actually have an additional adversary (the prover), and now require that $t_S + 1 < (n+1)/3$. Thus, depending on n, we can have different bounds on the maximum values of t_C and t_S and thus Γ_C may not be equal to Γ_S.

Protocol	Assumptions	Γ_C	Γ_S	Γ_Z
Protocol 0	Broadcast Channel	$\emptyset$	$\emptyset$	$\emptyset$
Protocol 0	no Broadcast Channel	$\mathcal{Q}_3$	$\mathcal{Q}_3$	$\emptyset$
Protocol 1	-	$\mathcal{Q}_3$	$\mathcal{Q}_3$	$\emptyset$
Protocol 2	Robust/identifiable abort MPC Protocol for Γ	Γ	Γ	Γ
Protocol 3	Threshold structures	$t_c < (n+1)/3$	$t_s < (n+1)/3 - 1$	$t_z < (n+1)/3$
$\Pi_{4\mathrm{t}}$	Digital Signatures	$t < n/4$	$t < n/4$	$t < n/4$
$\Pi_{3\mathrm{t}}$	Digital Signatures	$t < n/3$	$t < n/3$	$t < n/3$

Table 1. Comparison of Protocols

4 Preprocessing for distributed proofs with honest majority $t < n/2$

We begin by outlining the preprocessing phase for our proof in the presence of a honest majority. This preprocessing can then be used with the actual online phases of the proof, which require $t < n/4$ (Section 5) or $t < n/3$ (Section 6) corruptions. The ideal preprocessing functionality $\mathcal{F}_{\mathrm{Prep}}^{t,n}$ is described in Fig. 3. Both the protocols and functionality are defined over an extension field of appropriate degree to allow for Shamir secret sharing with n parties. We focus on the case of a binary field $\mathbb{F}_{2^k}$ with $2^k > n$, but our protocols are easily adapted to $\mathbb{F}_q$ for any $q > n$. We also use a repetition factor ρ such that $2^{k \cdot \rho} > 2^{\mathsf{sec}}$, where sec is our security parameter.

In the protocol $\Pi_{\mathsf{Prep}}^{\mathsf{t,n}}$ that implements the preprocessing functionality, and given in Fig. 4, each of the n verifiers $\mathcal{V}_i$ samples a random r_i and sends a share of $\langle r_i \rangle_t$ to each other verifier and r_i to the prover $\mathcal{P}$. These values are checked for consistency by forming a random linear combination using random values α_i. This random linear combination simultaneously guarantees the correctness of the underlying secret known to the prover and the consistency of the shares on a degree t polynomial. It can be repeated to achieve negligible soundness error. Next, let $\langle \vec{r} \rangle_t$ be the vector representing all sharings made by the verifiers, and let M_t be an $(n-t) \times n$ Vandermonde matrix. The verifiers locally compute the sharings $\langle \vec{s} \rangle_t = M_t \cdot \langle \vec{r} \rangle_t$, while the prover computes $\vec{s} = M_t \cdot \vec{r}$. This randomness extraction ensures that out of these n shares, of which t are known to the adversary, $n-t$ uniformly

Functionality $\mathcal{F}_{\text{Prep}}^{t,n}$

This functionality communicates with $n+1$ parties $\mathcal{P}, \mathcal{V}_1, \ldots, \mathcal{V}_n$ as well as the ideal adversary $\mathcal{S}$, where $\mathcal{P}$ denotes the prover and $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$ the verifiers. Let $\mathcal{W} = \mathcal{V} \cup \{\mathcal{P}\}$ and $t < n/2$.

Corrupt: Before any other command, $\mathcal{S}$ sends $(\mathsf{Corrupt}, \mathcal{D})$ where $\mathcal{D} \subseteq \mathcal{W}$. Let $\mathcal{H} = \mathcal{W} \setminus \mathcal{D}$. If $\mathcal{P} \in \mathcal{D}$ then we call the prover "corrupted", otherwise "honest". We call $\mathcal{V}_{\mathcal{D}} = \mathcal{V} \cap \mathcal{D}$ the corrupted verifiers and $\mathcal{V}_{\mathcal{H}} = \mathcal{V} \setminus \mathcal{V}_{\mathcal{D}}$ the honest verifiers.

Distribute Shares: On input (Shares, n_S) from all parties

1. Sample n_S random values $s_i \in \mathbb{F}_{2^k}$ for $i \in [n_S]$.
2. If $\mathcal{P}$ is corrupted then send $\{s_i\}_{i\in[n_S]}$ to $\mathcal{S}$.
3. Wait for a message (Abort, C_A) from $\mathcal{S}$ where $\emptyset \neq C_A \subseteq \mathcal{D}$ or $(\mathsf{Continue}, \{\hat{s}_i^{(p)}\}_{p\in\mathcal{V}_{\mathcal{D}}, i\in[n_S]})$.
   – If $\mathcal{S}$ inputs Abort then (Abort, C_A) is returned to each party in $\mathcal{H}$ and the functionality aborts.
   – If $\mathcal{S}$ inputs $\mathsf{Continue}$ then generate a Shamir sharing of s_i of degree t for each $i \in [n_H]$, which we denote by $\langle s_i \rangle_t$. The individual Shamir shares are denoted by $s_i^{(j)} \in \mathbb{F}_{2^k}$ for $j \in [n]$. The sharing is chosen so that $s_i^{(j)} = \hat{s}_i^{(j)}$. The values s_i are passed to $\mathcal{P}$ if $\mathcal{P} \in \mathcal{H}$, whilst the values $s_i^{(p)}$ are given to $\mathcal{V}_p$ for $p \in \mathcal{V}_{\mathcal{H}}$.

Fig. 3. Functionality $\mathcal{F}_{\text{Prep}}^{t,n}$ for preprocessing in the case when $t < n/2$

random shares are recovered, unknown to any other party than the prover. Several instances of this preprocessing phase are performed in parallel to obtain more than $n-t$ secret sharings, with (at least) an additional ρ sharings produced so as to verify the entire production is correct.

The protocol assumes a PKI in which each verifier $\mathcal{V}_i$ has a public key $\mathfrak{pk}_i$ and a signing key $\mathfrak{sk}_i$, which enables them to authenticate sent messages m with a digital signature $\mathsf{Sign}(\mathfrak{sk}_i, m)$. In the case when the consistency check fails, this allows parties to reveal the shares that they obtained from each other. This means that parties can identify cheaters by either identifying incorrectly generated sharings or incorrectly formed messages. Signatures prevent dishonest parties from framing honest parties by claiming to have obtained shares that the honest party never sent.

Theorem 1. *Assuming that* Sign *is an unforgeable signature scheme, then the protocol* $\Pi_{\mathsf{Prep}}^{\mathsf{t,n}}$ *in Fig. 4 securely implements the functionality* $\mathcal{F}_{\text{Prep}}^{t,n}$ *in the* $\mathcal{F}_{\text{Rand}}$*-hybrid model against any static adversary corrupting at most* $t < n/2$ *parties.*

Before proving the theorem, we give three lemmas that will simplify the proof. First, we show that if a dishonest party creates an incorrect sharing, then the protocol enters **Abort** with overwhelming probability. Second, we show that if a verifier sends an incorrect share to an honest prover, then the protocol enters **Abort** with overwhelming probability. Finally, we show that upon entering **Abort** at least one dishonest party is identified, and only dishonest parties are identified.

Lemma 2. *Let* $\mathcal{V}_{\mathcal{H}} = \mathcal{V} \cap \mathcal{H}$ *and assume* $t < n/2$*. For* $v \in [n]$*, consider the shares* $r_{v,j}^{(i)}$ *for* $\mathcal{V}_i \in \mathcal{V}_{\mathcal{H}}$ *and let* $S_{v,j}$ *be the unique polynomials of smallest degree over* $\mathbb{F}_{2^k}$ *such that* $S_{v,j}(i) = r_{v,j}^{(i)}$*. If there exist* v, j *such that*[7] $deg(S_{v,j}) > t$*, then the protocol enters* ***Abort*** *except with probability* $2^{-k\cdot\rho}$*.*

Proof. Computing $T_\ell^{(i)} = \sum_j \sum_v \alpha_{v,j,\ell} r_{v,j}^{(i)}$ is the same as computing the polynomials $S_\ell = \sum_{v,j} \alpha_{v,j,\ell} \cdot S_{v,j}$ first and then evaluating S_ℓ at points i to obtain the shares $T_\ell^{(i)}$ of the honest parties. This follows from the linearity of Lagrange interpolation.

[7] Here we use that $t < n/2$, as $S_{v,j}$ could otherwise not be of degree $> t$.

Protocol $\Pi^{\mathsf{t,n}}_{\mathsf{Prep}}$

We let M_t be an $(n-t) \times n$ Vandermonde matrix for randomness extraction. The protocol is parametrized by the number of verifiers n, number of corruptions $t < n/2$ and two integers n_S and ρ.
The protocol uses the hybrid functionality $\mathcal{F}_{\text{Rand}}$. If $\mathcal{F}_{\text{Rand}}$ sends (Abort, C_A) then each party in the protocol outputs (Abort, C_A) and terminates.

Distribute Shares:

1. Each party $\mathcal{V}_i \in \mathcal{V}$ executes the following protocol:
   (a) For $j \in [\lceil (n_S + \rho)/(n-t) \rceil]$ do
      i. Sample $r_{i,j} \in \mathbb{F}_{2^k}$ and generate a sharing $\langle r_{i,j} \rangle_t$.
      ii. Send $(r_{i,j}^{(p)}, \mathsf{Sign}(\mathfrak{sk}_i, r_{i,j}^{(p)}))$ to $\mathcal{V}_p$ for $p \neq i$. Note this is done as a single message for all j values needed.
      iii. Send $(r_{i,j}, \mathsf{Sign}(\mathfrak{sk}_i, r_{i,j}))$ to $\mathcal{P}$, again this is done as a single message for all j values needed.
      iv. On receiving $(r_{p,j}^{(i)}, \sigma_{p,j}^{(i)}) = (r_{p,j}^{(i)}, \mathsf{Sign}(\mathfrak{sk}_p, r_{p,j}^{(i)}))$ from party $\mathcal{V}_p$, verify the signature. If the signature $\sigma_{p,j}^{(i)}$ does not hold or if $\mathcal{V}_p$ did not send any message at all
         A. Broadcast $(\mathsf{Complaint}, i, \mathcal{V}_p)$.
         B. Upon receiving $(\mathsf{Complaint}, i, \mathcal{V}_p)$ party $\mathcal{V}_p$ publicly sends $(r_{p,j}^{(i)}, \sigma)$ to all parties, who forward it to $\mathcal{V}_i$.
      v. Similarly, do the same for the signatures that $\mathcal{P}$ should obtain.
2. For $\ell \in [\rho]$ do as follows.
   (a) Execute $(\alpha_{1,j,\ell}, \dots, \alpha_{n,j,\ell}) \leftarrow \mathcal{F}_{\text{Rand}}(\{\mathcal{V}_1, \dots, \mathcal{V}_n, \mathcal{P}\}, n, \mathbb{F}_{2^k})$.
   (b) Compute $T_\ell^{(i)} \leftarrow \sum_j \sum_{v \in [n]} \alpha_{v,j,\ell} \cdot r_{v,j}^{(i)}$ and broadcast $T_\ell^{(i)}$.
   (c) The prover $\mathcal{P}$ computes $T_\ell \leftarrow \sum_j \sum_{v \in [n]} \alpha_{v,j,\ell} \cdot r_{v,j}$ and broadcasts T_ℓ.
   (d) If the $T_\ell^{(i)}$ do not form a valid degree-t sharing of T_ℓ then go to **Abort(ℓ)**.
3. For $j \in \lceil n_S/(n-t) \rceil$ do
   (a) $c \leftarrow (j-1) \cdot (n-t)$.
   (b) The prover $\mathcal{P}$ computes and outputs $(s_{1+c}, \dots, s_{n-t+c})^T = M_t \times (r_{1,j}, \dots, r_{n,j})^T$,
   (c) $\mathcal{V}_i \in \mathcal{V}$ compute and output $(\langle s_{1+c} \rangle_t, \dots, \langle s_{n-t+c} \rangle_t)^T = M_t \times (\langle r_{1,j} \rangle_t, \dots, \langle r_{n,j} \rangle_t)^T$.

Abort(ℓ): Each $\mathcal{V}_i$ holds $r_{v,j}^{(i)}, \sigma_{v,j}^{(i)}$ for $v \in [n]$ and $j \in [\lceil (n_S + \rho)/(n-t) \rceil]$, while $\mathcal{P}$ holds $r_{v,j}, \sigma_{v,j}$ for $v \in [n]$ and $j \in [\lceil (n_S + \rho)/(n-t) \rceil]$ (for simplicity, each $\mathcal{V}_i$ signs a share $r_{i,j}^{(i)}$ for itself).

1. Each verifier $\mathcal{V}_i$ broadcasts $\{r_{v,j}^{(i)}, \sigma_{v,j}^{(i)}\}_{v,j}$, while $\mathcal{P}$ broadcasts $\{r_{v,j}, \sigma_{v,j}\}_{v,j}$. If any signature $\sigma_{v,j}^{(i)}$ does not hold then identify $\mathcal{V}_i$ as a cheater and abort. If any $\sigma_{v,j}$ does not hold then identify $\mathcal{P}$ as cheater and abort.
2. If for some $i \in [n]$ it holds that $T_\ell^{(i)} \neq \sum_{v,j} \alpha_{v,j,\ell} \cdot r_{v,j}^{(i)}$ then identify $\mathcal{V}_i$ as cheater and abort. If it holds that $T_\ell \neq \sum_j \sum_v \alpha_{v,j,\ell} \cdot r_{v,j}$ then identify $\mathcal{P}$ as a cheater and abort.
3. For any $\mathcal{V}_v$, if $r_{v,j}^{(1)}, \dots, r_{v,j}^{(n)}$ do not form a valid degree-t sharing of $r_{v,j}$ then identify $\mathcal{V}_v$ as a cheater and abort.

Fig. 4. Protocol for preprocessing with $t < n/2$

Any additional point $T_\ell^{(v)}$ provided by the adversary through party $\mathcal{V}_v$ can either lie on the polynomial S_ℓ or not. If it does then S_ℓ will keep its degree, if not then the points $T_\ell^{(1)}, \dots, T_\ell^{(n)}$ must lie on a polynomial of larger minimal degree. This means that the protocol enters **Abort** if any of the protocols S_ℓ is of degree $> t$, independent of the values $T_\ell^{(v)}$ sent by $\mathcal{S}$.

Let $r = \max_{v,j}\{\deg(S_{v,j})\}$, by definition we have $r > t$. This means that for some $S_{v,j}$ the monomial X^r has a non-zero coefficient. Then any S_ℓ will only be of degree $< r$, i.e. the shares of honest parties will lie on a degree-$< r$ polynomial, if the coefficients of the monomials X^r of all $S_{v,j}$ sum to 0 in S_ℓ. By the random choice of the $\alpha_{v,j,\ell}$ through $\mathcal{F}_{\text{Rand}}$ after these $S_{v,j}$ are fixed, this only happens with probability 2^{-k} for a single S_ℓ and with probability $2^{-k\rho}$ for all $S_1, \ldots, S_\rho$ simultaneously. □

Lemma 3. *Let $\mathcal{V}_\mathcal{H} = \mathcal{V} \cap \mathcal{H}$ and $t < n/2$ and assume $\mathcal{P} \in \mathcal{H}$. For $v \in [n]$, consider the shares $r_{v,j}^{(i)}$ of $\mathcal{V}_i \in \mathcal{V}_\mathcal{H}$ and let $S_{v,j}$ be the unique polynomials of degree t over $\mathbb{F}_{2^k}$ such that $S_{v,j}(i) = r_{v,j}^{(i)}$. Furthermore, let $r_{v,j}$ be the values received by $\mathcal{P}$. If there exist v, j such that $S_{v,j}(0) \neq r_{v,j}$, then the protocol enters* ***Abort*** *except with probability $2^{-k \cdot \rho}$.*

Proof. Observe that $\alpha_{v,j,\ell}$ are only chosen through $\mathcal{F}_{\text{Rand}}$ after all $r_{v,j}^{(i)}, r_{v,j}$ have been fixed, $v \in [n]$.

Assume that the protocol does not enter **Abort**, then for each $\ell \in [\rho]$ it holds that

$$\sum_{v,j} \alpha_{v,j,\ell} r_{v,j} = \sum_{v,j} \alpha_{v,j,\ell} \cdot S_{v,j}(0)$$

which can be rewritten as

$$0 = \sum_{v,j} \alpha_{v,j,\ell} \cdot (r_{v,j} - S_{v,j}(0))$$

Write $S_{v,j}(0) = r_{v,j} + \delta_{v,j}$. By assumption, there must exist v, j such that $\delta_{v,j} \neq 0$. Hence it must hold that the $\delta_{v,j}$ chosen by the adversary lie in the kernel of $\alpha_{v,j,\ell}$ which are chosen uniformly at random after $\delta_{v,j}$ are fixed. For any ℓ, this happens with probability at most 2^{-k} and with probability at most $2^{-k\rho}$ for all $\ell \in [\rho]$ simultaneously. □

Lemma 4. *Assuming unforgeability of* Sign*, then* ***Abort*** *always terminates with at least one dishonest party being identified. Furthermore, it only terminates identifying dishonest parties.*

Proof. In Step 1 of **Abort** the protocol only identifies dishonest parties. This is because honest parties would have asked for shares with valid signatures in Step 1(a)iv of **Distribute Shares**. Similarly, we identify a dishonest prover as an honest prover would have asked for correctly signed data in Step 1(a)v of **Distribute Shares**.

In Step 2 we only identify dishonest parties, as honest parties would have computed $T_\ell^{(i)}, T_\ell$ correctly.

Assuming we reach Step 3 without aborting, then all $T_\ell^{(i)}, T_\ell$ were computed correctly but either $T_\ell^{(i)}$ do not form a polynomial of degree t or do not share the secret T_ℓ. If for each v, j the shares $r_{v,j}^{(i)}$ would form a degree-t sharing of $r_{v,j}$ then the condition for entering **Abort** cannot be reached. Thus, there must exist v, j such that the polynomial formed by $r_{v,j}^{(i)}$ is of larger degree or reconstructs to a value that is not $r_{v,j}$.

If $\mathcal{V}_v$ was honest then all $r_{v,j}^{(1)}, \ldots, r_{v,j}^{(n)}$ revealed during Step 1 lie on a degree-t polynomial. The protocol only identifies an honest party $\mathcal{V}_v$ in Step 3 if $r_{v,j}^{(1)}, \ldots, r_{v,j}^{(n)}$ lie on a polynomial of degree $t+1$ or higher. As honest parties report the shares of $\mathcal{V}_v$ honestly, this only happens if an incorrect $\tilde{r}_{v,j}^{(i)}$ is broadcast by a corrupt $\mathcal{V}_i$, together with a valid signature under $\mathfrak{sk}_v$ (as we would have otherwise aborted in Step 1). So an honest $\mathcal{V}_v$ is only identified as a cheater if a signature was forged by $\mathcal{V}_i$, contradicting the assumption that the signature scheme is unforgeable. Similarly, an honest $\mathcal{V}_v$ would always send the correct shared $r_{v,j}$ to $\mathcal{P}$ so $\mathcal{P}$ can only reveal $\tilde{r}_{v,j}$ that is inconsistent with $r_{v,j}^{(1)}, \ldots, r_{v,j}^{(n)}$ if it can forge a signature, contradicting the assumption. Therefore, any $\mathcal{V}_v$ identified by Step 3 must be corrupted. □

We can now give the simulation-based proof of Theorem 1.

Proof. (of Theorem 1) The simulator $\mathcal{S}$ obtains as input from the environment the set $\mathcal{D}$ of corrupted parties and forwards this to $\mathcal{F}_{\text{Prep}}^{t,n}$. It furthermore sets up a copy of $\mathcal{F}_{\text{Rand}}$. If $\mathcal{P} \in \mathcal{H}$ then $\mathcal{S}$ will simulate an honest prover. Moreover, for each $\mathcal{V}_i \in \mathcal{H}$ $\mathcal{S}$ will simulate an honest verifier. It will generally follow the protocol, except if specified otherwise below. Initially, let $C_A = \emptyset$. Send (Shares, n_S) in the name of all simulated honest parties to $\mathcal{F}_{\text{Prep}}^{t,n}$. If $\mathcal{P} \in \mathcal{D}$ then $\mathcal{S}$ obtains the shares s_i from $\mathcal{F}_{\text{Prep}}^{t,n}$. If at any point $\mathcal{F}_{\text{Rand}}$ outputs (Abort, C_A) then $\mathcal{S}$ sends (Abort, C_A) to $\mathcal{F}_{\text{Prep}}^{t,n}$.

$\mathcal{S}$ simulates the honest verifiers $\mathcal{V}_i$ in Step 1(a)ii by sending uniformly random $r_{i,j}^{(v)}$ to each corrupted $\mathcal{V}_v$. It then waits for the sharings of the dishonest parties being sent to the simulated honest verifiers. If any of these sharings is of degree $> t$ for a dishonest verifier $\mathcal{V}_v$ then add v to the set C_A, otherwise denote $\langle r_{v,j} \rangle_t$ as the secret sharings of the dishonest parties.

If $\mathcal{P} \in \mathcal{D}$ then choose $r_{i,j}$ for the honest verifiers such that a prover following Step 3 will obtain s_i as output, and send these $r_{i,j}$ to the corrupt $\mathcal{P}$. This is always possible using [BTH08]. If instead $\mathcal{P} \notin \mathcal{D}$ then choose uniformly random $r_{i,j}$ for each honest verifier and wait for values $\tilde{r}_{v,j}$ being sent from the dishonest verifiers to the simulated $\mathcal{P}$. For any of these shares $\langle r_{v,j} \rangle_t$ that does not reconstruct to $\tilde{r}_{v,j}$ add v to C_A. Finally, choose suitable $r_{i,j}^{(p)}$ for all honest $\mathcal{V}_p$ to create valid sharings $\langle r_{i,j} \rangle_t$.

If the protocol enters **Abort**, then $\mathcal{S}$ follows **Abort** honestly but aborts the simulation when a dishonest party provides a forged signature in Step 1 of **Abort**. Additionally, it adds to C_A any dishonest party that sent incorrect $T_\ell^{(i)}$ or T_ℓ if $\mathcal{P} \in \mathcal{D}$, as identified in **Abort**.

If $C_A \neq \emptyset$ then $\mathcal{S}$ sends (Abort, C_A) to $\mathcal{F}_{\text{Prep}}^{t,n}$, independent if **Abort** of the protocol was entered or not. Otherwise it computes $\langle s_i \rangle_t$ as parties would do in the protocol and sends the shares of the dishonest parties to $\mathcal{F}_{\text{Prep}}^{t,n}$.

Indistinguishability. We first observe that the shares of the honest parties which the environment obtains from $\mathcal{F}_{\text{Prep}}^{t,n}$ are consistent with those of the dishonest parties if the simulation finishes successfully. This is because if $\mathcal{P}$ is corrupted then the shares will be consistent with the s_i, while they are otherwise consistent with the s_i unknown to the adversary during the protocol run as the adversary does not have enough shares to reconstruct (and $\mathcal{F}_{\text{Prep}}^{t,n}$ chooses the shares of the honest parties accordingly). Moreover, $\mathcal{S}$ always aborts $\mathcal{F}_{\text{Prep}}^{t,n}$ if the adversary provides inconsistent shares to honest parties or if they provably send visibly incorrect $T_\ell^{(i)}, T_\ell$. We now show through a sequence of hybrids that the output of $\mathcal{S}$ when interacting with the dishonest parties is indistinguishable from the real protocol running with the dishonest parties.

Define the output of the simulation as H_0 and let H_1 be exactly like H_0, but where dishonest $\mathcal{V}_v$ that send invalid $r_{v,j}$ to an honest $\mathcal{P}$ are only added to C_A if the protocol actually enters **Abort**. By Lemma 3, these two hybrids are indistinguishable except with probability $2^{-k\rho}$.

Let H_2 be the same hybrid as H_1, but where dishonest $\mathcal{V}_v$ are only added to C_A if they were identified to have sent incorrect sharings in **Abort**. By Lemma 2, these two hybrids are indistinguishable except with probability $2^{-k\rho}$.

Observe that in the computation of C_A, only dishonest parties are contained and the simulation would abort. Now, let H_3 be the same as H_2 but where the simulation does not abort. As abort of the simulation happens iff the adversary succeeds in forging a signature, any distinguisher of H_2 and H_3 can be used to successfully break the unforgeability of Sign. Finally, observe that the

Protocol $\Pi_{4\text{t}}$

Let C be the circuit to be proved; the prover $\mathcal{P}$ is assumed to know an input witness w such that $C(w) = 0$. Let n_S denote the number of AND gates in the circuit, n_W the length of the witness w and ρ a positive integer.
Let $\mathsf{CheckMult}$ and $\mathsf{OutputRec}$ be two additional flags initially set to $\top$ and $\bot$ respectively.

Init: Call $\mathcal{F}^{t,n}_{\text{Prep}}$, so that $\mathcal{P}$ obtains s_i and the verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ obtain $\langle s_i \rangle_t$ for $i \in [n_S + n_W + 3\rho]$, i.e. $\mathcal{V}_j$ obtains $s_i^{(j)}, j \in [n]$. Set $a_j = s_{j+n_W+n_s+\rho}$ and $b_j = s_{j+n_W+n_s+2\rho}, j \in [\rho]$.

Prove: The prover "evaluates" the circuit as follows:

1. Compute the difference between the input wire values w_i and the pre-processed values s_i, i.e. $w_i - s_i, i \in [n_W]$.
2. Evaluate the circuit gate-by-gate:
   (a) For every linear gate, simply compute the resulting wire value
   (b) For each AND gate, compute the resulting wire value $z_j \leftarrow x_j \cdot y_j$ and $z_j - s_{j+n_W}, j \in [n_S]$.
   (c) Compute ρ additional random triples as $a_j \cdot b_j = c_j$, and $c_j - s_{j+n_W+n_S}$, $j \in [\rho]$
3. Set the proof to be the concatenation of all the values $\{w_i - s_i\}_{i\in[n_W]}$, $\{z_j - s_{j+n_W}\}_{j\in[n_S]}$, and $\{c_j - s_{j+n_W+n_S}\}_{j\in[\rho]}$.

Verify: The verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ jointly check the circuit evaluation:

1. Evaluate the circuit within the Shamir secret sharing, computing a share of the output wire $\langle o \rangle_t$:
   (a) Shares of the input wires can be computed as $\langle w_i \rangle_t \leftarrow \langle s_i \rangle_t + (w_i - s_i)$ for $i \in [n_W]$.
   (b) Shares of the output wire values for an AND gate can be computed as
   $$\langle c_j \rangle_t \leftarrow \langle s_{j+n_W} \rangle_t + (c_j - s_{j+n_W}), \quad \text{for} \;\; j \in [n_S].$$
   (c) A degree $2 \cdot t$ sharing $\langle c_j \rangle_{2\cdot t}$ of this same value is computed by each $\mathcal{V}_i$ as $c_j^{(i)} \leftarrow a_j^{(i)} \cdot b_j^{(i)}$.
   (d) Linear gates can be evaluated linearly over the shares in the degree t sharing.
2. The verifiers call $\mathsf{RobustReconstruct}(\langle o \rangle_t, t)$, to obtain (o, flag_o).
3. If $o \neq 0$:
   – If $\mathsf{flag}_o = (\mathsf{correct}, \emptyset)$ then output Fail.
   – If $\mathsf{flag}_o = (\mathsf{incorrect}, C_o)$, then output the dishonest verifiers in C_o and Fail.
4. Else, set $\mathsf{OutputRec} = \top$. If $\mathsf{flag}_o = (\mathsf{incorrect}, C_o)$, identify the dishonest verifiers in C_o.
5. *Multiplications check:* Verifiers repeat ρ times the following.
   (a) Call $(\beta_1, \ldots, \beta_{n_S+1}) \leftarrow \mathcal{F}_{\text{Rand}}(\{\mathcal{V}_1, \ldots, \mathcal{V}_n\}, n_S + 1, \mathbb{F}_{2^k})$.
   (b) Compute
   $$\langle A \rangle_{2t} = \sum_{j\in[n_S]} \beta_j \cdot \langle z_j \rangle_{2t} + \beta_{n_S+1} \cdot \langle c_i \rangle_{2t} \;\; \text{and} \;\; \langle C \rangle_t = \sum_{j\in[n_S]} \beta_j \cdot \langle z_j \rangle_t + \beta_{n_S+1} \cdot \langle c_i \rangle_t$$
   (c) Run $\mathsf{RobustReconstruct}(\langle A \rangle_{2t} - \langle C \rangle_t, t)$, to obtain (T, flag_T)
   (d) If $T \neq 0$, set $\mathsf{CheckMult} = \bot$. Moreover,
      – If $\mathsf{flag}_T = (\mathsf{correct}, \emptyset)$, then output Fail.
      – If $\mathsf{flag}_{T_v} = (\mathsf{incorrect}, C_M)$, then identify the dishonest verifiers in flag_{T_v} and output Abort
6. If both $\mathsf{CheckMult} = \top$ and $\mathsf{OutputRec} = \top$, accept the proof and identify possible dishonest verifiers $C_A = C_o \cup \{C_{M_v}\}_{v\in[\rho]}$

Fig. 5. Protocol $\Pi_{4\text{t}}$ for $t < n/4$

distribution of the shares of the honest parties, the identified corrupted parties as well as the abort events are identical between H_3 and the protocol. □

5 Distributed proof with $t < n/4$ corruptions

In this section we describe a protocol which deals with $t < n/4$ corruptions of the verifiers, i.e. Γ_C, Γ_S and Γ_Z are access structures consisting of all sets with more than $n - t$ verifiers in them. The protocol $\Pi_{4\mathrm{t}}$, given in Fig. 5, forms the basis of our following protocol in the case of $t < n/3$, indeed it shares the same pre-processing phase from the previous section.

In the setting where we have $t < n/4$ corruptions we can rely on the Reed-Solomon decoding to robustly open secret sharings of degree up to $2t$. Thus we can efficiently verify multiplications. We assume the statement to be verified is given by a circuit C over $\mathbb{F}_{2^k}$ which will evaluate to zero only on input of the witness w, i.e. $C(w) = 0$.

Given the values $\vec{s}$ generated in pre-processing, the prover can trivially "commit" to the witness w as well as the outputs of all the multiplication gates of C by broadcasting the difference between $\vec{s}$ and these values towards the verifiers. The verifiers can then evaluate the circuit as follows: to obtain the wire output values of a gate, they can either simply apply the corresponding linear operation directly on their shares, or obtain a sharing for the output wire from the prover's broadcast for multiplications. After evaluating the entire circuit in this manner, the verifiers can robustly open $\langle C(w)\rangle_t$ and verify it correctly evaluates to zero.

The verifiers also have to check that the commitments the prover provided for the outputs of the multiplication gates are consistent. For each verifier $\mathcal{V}_i$, let $a_j^{(i)}$ be the share of the left input corresponding to the jth multiplication/AND gate, $j \in [n_S]$. Correspondingly, $b_j^{(i)}$ is the share for the right input and $c_j^{(i)}$ for the output. Then $c_j^{(i)} = a_j^{(i)} \cdot b_j^{(i)}$ is a degree $2 \cdot t$ sharing of the value $c_j = a_j \cdot b_j$ output by this multiplication gate. We represent this sharing by $\langle c_j\rangle_{2\cdot t}$. The proof proceeds by verifying that the values held in $\langle c_j\rangle_{2\cdot t}$ are identical with the values held in $\langle c_j\rangle_t = \langle s_j\rangle_t - (s_j - c_j)$, and provided by the prover, therefore checking that all committed multiplication gate outputs were correct.

To achieve this, the verifiers check that a random linear combination over all products of the inputs corresponds to the same linear combination over the gate outputs. More precisely, for each multiplication gate $j \in [n_S]$, the verifiers sample a uniformly random multiplier β_j and locally compute shares $A^{(i)} = \sum_j \beta_j \cdot a_j^{(i)} \cdot b_j^{(i)}$, and $C^{(i)} = \sum_j \beta_j \cdot c_j^{(i)}$. Then, since $t < n/4$, the verifiers reliably reconstruct $\langle A\rangle_{2t}$ and $\langle C\rangle_t$. If $A = C$ then the verifiers accept the proof, otherwise they reject. Cheater identification can be achieved in a straightforward manner thanks to the error correction during the robust reconstruction. Moreover, the check is made zero-knowledge by letting $\mathcal{P}$ share additional valid random multiplication triples.

Theorem 2. *If $t < n/4$ then protocol $\Pi_{4\mathrm{t}}$ secure implements the functionality $\mathcal{F}_{\mathrm{DV-ZK}}$ in the $(\mathcal{F}_{\mathrm{Prep}}^{t,n}, \mathcal{F}_{\mathrm{Rand}})$-hybrid model with $\Gamma_C = \Gamma_S = \Gamma_Z$ being the set of all subsets of verifiers of size $n-t$ or more.*

In the proof, we use the following lemma.

Lemma 5. *Let $\langle x_j\rangle_t, \langle y_j\rangle_t, \langle z_j\rangle_{2t}, \langle z_j\rangle_t, \langle a\rangle_t, \langle b\rangle_t, \langle c\rangle_{2t}, \langle c\rangle_t$ the inputs of the multiplications check. If either $x_j \cdot y_j \neq z_j$, for some $j \in [n_S]$, or $a \cdot b \neq c$, then $T \neq 0$, except with probability $\frac{1}{|\mathbb{F}|}$.*

Proof. (of Lemma 5) We recall that $\langle z_j\rangle_t = \langle s_j\rangle_t - (s_j - z_j)$, $j \in [n_S]$, and $\langle c\rangle_t = \langle s\rangle_t - (s - c)$, where $\langle s_j\rangle_t$ and $\langle s\rangle_t$ are correct sharings provided by the preprocessing functionality. Let

$f_{j,t}(\cdot), g_{j,t}(\cdot), s_{j,t}(\cdot)$ be the unique t-degree polynomials such that, for $j \in [n_S]$,

$$\begin{aligned} f_{j,t}(i) &= x_j^{(i)}, \quad f_{j,t}(0) = x_j, \\ g_{j,t}(i) &= y_j^{(i)}, \quad g_{j,t}(0) = y_j, \\ s_{j,t}(i) &= s_j^{(i)}, \quad s_{j,t}(0) = s_j, \end{aligned}$$

and $p_t(\cdot), q_t(\cdot), s_t(\cdot)$ the unique t-degree polynomials such that

$$\begin{aligned} p_t(i) &= a^{(i)}, \quad p_t(0) = a. \\ q_t(i) &= b^{(i)}, \quad q_t(0) = b, \\ s_t(i) &= s^{(i)}, \quad s_t(0) = s. \end{aligned}$$

Then the shares $A^{(i)}$ and $C^{(i)}$ are given by

$$\sum_{j\in[n_S]} \beta_j \cdot (f_{j,t}(i) \cdot g_{j,t}(i)) + \beta_{n_S+1} \cdot (p_t(i) \cdot q_t(i))$$

and

$$\sum_{j\in[n_S]} \beta_j \cdot (s_{j,t}(i) - (s_j - z_j)) + \beta_{n_S+1} \cdot (s_t(i) - (s - c)).$$

If all the triples are correct, then $A - C = T = 0$.

Otherwise, suppose that $f_{j,t}(0)\cdot g_{j,t}(0) = \tilde{z}_j$ and $p_t(0)\cdot q_t(0) = \tilde{c}$ with $\tilde{z}_j = z_j+\delta_j$ and $\tilde{c} = c+\delta_{n_S+1}$. Then the reconstructed value T is given by

$$\begin{aligned} A - C &= \sum_{j\in[n_S]} \beta_j(\tilde{z}_j - z_j) + \beta_{n_S+1}(\tilde{c} - c) \\ &= \sum_{j\in[n_S]} \beta_j \cdot \delta_j + \beta_{n_S+1} \cdot \delta_{n_S+1}, \end{aligned}$$

where not all δ_j's are zero. Let $\vec{b} = (\beta_1, \ldots, \beta_{n_S}, \beta_{n_S+1})$ and $\vec{d} = (\delta_1, \ldots, \delta_{n_S}, \delta_{n_S+1})$, and consider the liner map $f_d = \vec{d}\cdot\vec{b}^T$. The probability that T is zero is equal to the probability that $\vec{b} \in \ker(f_d)$. Since $\dim(\ker(f_d)) = n_S$, and $\vec{b}$ is random and unknown to $\mathcal{A}$ when they choose $\vec{d}$, the probability that $\vec{b} \in \ker(f_d)$ is $\frac{|\mathbb{F}|^{n_S}}{|\mathbb{F}|^{n_S+1}} = \frac{1}{|\mathbb{F}|}$. $\square$

Proof. (of Theorem 2) The simulator $\mathcal{S}$ obtains as input from the environment the set $\mathcal{D}$ of corrupted parties and forwards $(\mathsf{Corrupt}, \mathcal{D})$ to the functionality. On input (Init) from $\mathcal{F}_{\mathrm{DV-ZK}}$, if $\mathcal{A}$ sends Abort, it forwards Abort to the functionality, otherwise it forwards (ok). $\mathcal{S}$ sets up a copy of $\mathcal{F}_{\mathrm{Rand}}$.

$\mathcal{S}$ emulates $\mathcal{F}^{t,n}_{\mathrm{Prep}}$ obtaining s_i and the $s_i^{(j)}$, for $i \in [n_S+n_W+3\rho]$, held by the corrupted parties. Since $\mathcal{V}_{\mathcal{D}} \in \Delta_Z$, it receives (Prove, x) from the functionality. If $\mathcal{P} \in \mathcal{H}$, then it randomly samples the shares of the proof for corrupted parties, and sets honest shares consistently, i.e., such that the multiplication values are correct and $o = 0$; otherwise it receives from $\mathcal{A}$ the proof, consisting of the masked input values and masked multiplication values for AND gates and ρ masked random triples. In this case, $\mathcal{S}$ reconstructs the input $\tilde{w}$ and forwards $(\mathsf{Prove}, x, \tilde{w})$ to the functionality. The simulator $\mathcal{S}$ starts the simulation of the verification step, i.e. it evaluates the circuit honestly, for each gate computes the shares held by corrupted parties and sends the shares of the honest parties needed to run $\mathsf{RobustReconstruct}(\langle o \rangle_t, t)$. Receiving the shares of corrupted parties, it checks if those shares are the same as the ones computed by $\mathcal{S}$. We distinguish two different cases:

– If $\mathcal{P} \in \mathcal{H}$: The simulator $\mathcal{S}$ sets the flag accept. If the shares are consistent then $C_A = \emptyset$; else, if the shares are inconsistent, it identifies the cheating verifiers with incorrect shares and updates C_A with those parties.
– If $\mathcal{P} \notin \mathcal{H}$: if either $(x, w) \notin \mathcal{R}$ or some of the multiplication values given by the prover are incorrect, it sets the flag reject. If some of the shares are inconsistent, then $\mathcal{S}$ identifies the cheaters and updates C_A.

After this, $\mathcal{S}$ emulates the Multiplications check. To do this, it obtains random $\beta_1, \ldots, \beta_{n_S+1}$ from $\mathcal{F}_{\text{Rand}}$, and sends these values to $\mathcal{A}$. If at any time $\mathcal{F}_{\text{Rand}}$ sends (Abort, C_A), the simulator forwards (Abort, C_A) to the functionality. It also sends to $\mathcal{A}$ the honest shares $A^{(i)}, C^{(i)}, i \in \mathcal{H}$, necessary to run $\mathsf{RobustReconstruct}$, and receives from $\mathcal{A}$ the values $A^{(j)}, C^{(j)}, j \in \mathcal{D}$. If some of these shares are incorrect, it updates C_A with the corresponding corruptions.

Finally, if the flag accept or reject is true, $\mathcal{S}$ sends $(\mathsf{Abort}, 1, C_A)$ to $\mathcal{F}_{\text{DV}-\text{ZK}}$, otherwise it sends $(\mathsf{Abort}, 0, C_A)$ to the functionality.

Indistinguishability. We now argue indistinguishability of the real and ideal executions to an environment, $\mathcal{Z}$. Recall that $\mathcal{Z}$ chooses the inputs of all parties. The view of $\mathcal{Z}$ in the real world then consists of these inputs, the messages received by the adversary and all the output values.

Indistinguishability of the proof follows from the privacy of Shamir's secret sharing scheme and from the fact that the input and the multiplication values are masked by the preprocessed values s_i, that are unknown to the adversary if the prover is honest. The messages received by the adversary in the multiplications check are randomized by a triple a, b, c, different for each of the ρ executions and randomly chosen by the simulator, if $\mathcal{P}$ is honest, and unknown to $\mathcal{Z}$. From this and privacy of Shamir's sharings, simulation of these messages is perfect.

To argue indistinguishability of the output, we distinguish two cases as follows.

– If $\mathcal{P} \in \mathcal{H}$, the simulator always accepts the proof and outputs $(\mathsf{Abort}, 1, C_A)$ to the functionality, where the set $C_A = \emptyset$ if all the shares provided by $\mathcal{A}$ are correct and consistent. Irrespective of what the adversary does, $\mathsf{RobustReconstruct}$ always reconstructs the correct values, even with $\mathsf{flag} = (\mathsf{incorrect}, C)$, since $t < n/4$. In the ideal execution, the simulator outputs the set of parties that provided incorrect shares, in the real one this same set is provided by $\mathsf{RobustReconstruct}$. Indeed, since the sharing is correct, it is possible to efficiently and correctly detect all the $t < n/4$ possible errors. Hence, in this case the simulation is perfect.
– If $\mathcal{P} \notin \mathcal{H}$, the simulator honestly evaluates the circuit with inputs extracted from the masked proof given by $\mathcal{A}$. Therefore, if $(x, \tilde{w}) \notin \mathcal{R}$ and the multiplication values that are part of the proof are correct, then $\mathcal{S}$ rejects the proof by sending $(\mathsf{Abort}, 1, C_A)$ and the outputs of the two executions are identical.
  If $(x, \tilde{w}) \notin \mathcal{R}$ and the multiplication values are incorrect, then the simulator rejects the proof by sending $(\mathsf{Abort}, 1, C_A)$, whereas in the real execution the probability of acceptance is given by Lemma 5.
  Since this test is repeated ρ times, the probability of passing the multiplications check with incorrect inputs is $(\frac{1}{|\mathbb{F}|})^\rho$. Finally, we note that also in this case the set C_A of corrupted verifiers

given by the simulation and the protocol are identical and only consists of dishonest parties: while $\mathcal{S}$ can directly check inconsistent shares, in a real execution this set is guaranteed to be correct by the correctness of the sharing provided by $\mathcal{F}_{\text{Prep}}^{t,n}$.

□

6 Distributed Proof with $t < n/3$ corruptions

The general approach for this setting will be very similar to the case $t < n/4$ described in the previous section. The main difference is that now we can no longer robustly reconstruct a degree $2t$ polynomial, so we will instead rely on the Schwartz-Zippel lemma to check the correctness of the multiplications. More precisely, we use a checking method similar to the one used [BBC+19, BdK+21]. We split the n_S total multiplication gate into n_1 batches of n_2 gates each (i.e. $n_S = n_1 \cdot n_2$). For each batch j of n_2 gates, let the n_2 left inputs to the gate be $a_{j,k}$ and the right inputs be $b_{j,k}$, for $k \in [n_2]$. The prover then samples and shares $2 \cdot n_1$ random field elements t_i, in the usual way by computing the difference with fresh preprocessing elements. Once the prover is committed to all values $a_{j,k}, b_{j,k}, a_{j,k} \cdot b_{j,k}, t_i$, they sample n_1 random field elements r_i, seeded with those commitment values, using the Fiat-Shamir transform.

The prover proceeds by computing the polynomial

$$P(x) = \sum_{j \in [n_1]} A_j(x) \cdot B_j(x),$$

where $A_j(k) = r_j \cdot a_{j,k}$, $B_j(k) = b_{j,k}$, for $k \in [n_1]$, and $A(n_2) = t_{2 \cdot j-1}$, $B(n_2) = t_{2 \cdot j}$, interpolated for every individual batch of multiplication gates with fresh random elements t_i to mask the linear combinations the verifiers will reveal. Notice $P(k) = \sum_{j \in [n_1]} r_j \cdot (a_{j,k} \cdot b_{j,k})$ when $k \in [n_1]$, and hence the verifiers can already compute these values as a linear combination of the circuit wires. By sharing $n_2 + 2$ more values $P(k)$, the verifiers can then interpolate inside the secret sharing to obtain a secret sharing of $P(x)$. Finally, to verify the proof, the verifiers can check that the circuit evaluates to 0, as in the protocol for $t < n/4$. Here they first sample a uniformly random $\zeta \in \mathbb{F}$ and apply the Schwartz-Zippel lemma to check that $P(\zeta) = \sum_{j \in [n_1]} A_j(\zeta) \cdot B_j(\zeta)$ by computing and robustly opening $\langle P(\zeta) \rangle_t$, $\langle A_j(\zeta) \rangle_t$ and $\langle B_j(\zeta) \rangle_t$. See Fig. 6 for a full formal description of the protocol.

In contrast to the case for $t < n/4$, we not only have the constraint that $|\mathbb{F}| > n$ from the need for the Shamir sharing among the verifiers, but we also need to ensure that the interpolation of the checking polynomials $A_j(x)$, $B_j(x)$ and $P(x)$ is possible. This means that we require $|\mathbb{F}| > 2 \cdot n_2$. For the Schwartz-Zippel lemma, the soundness error is bounded by $\frac{2 \cdot n_2}{|\mathbb{F}| - n_2}$, and the total soundness error of the approach is bounded by $\frac{1}{|\mathbb{F}|} + \frac{2 \cdot n_2}{|\mathbb{F}| - n_2} \leq \frac{2 \cdot n_2 + 1}{|\mathbb{F}| - n_2}$. This error can be made smaller by making $\mathbb{F}$ large enough such that $|\mathbb{F}| \geq 2^{\mathsf{sec}} \cdot (2 \cdot n_2 + 1)$, which requires $\mathsf{sec} + \lceil \log_2(2 \cdot n_2 + 1) \rceil$ bits of communication per party to open any value. Alternatively, we can perform $\rho = \lceil \frac{\mathsf{sec}}{\log_2(|\mathbb{F}|) - \log_2(2 \cdot n_2 + 1)} \rceil$ parallel repetitions, which then requires $\rho \cdot (\lceil \log_2(n_2) \rceil + 1)$ bits of communication per party. To also take into account the number q of queries to $\mathcal{H}$ we allow the prover to make, a more accurate requirement is that $\left(\frac{q}{|\mathbb{F}|} + \frac{2 \cdot n_2}{|\mathbb{F}| - n_2} \right)^{\rho} \leq 2^{-\mathsf{sec}}$. This can be bounded by $q \cdot \epsilon$ for easier parameter selection when the prover has to commit to all ρ instances simultaneously when sampling all t_i, if we let ϵ be the total soundness error. The protocol given in Fig. 6, represents a single instance of the procedure described above, and can be used when the field size is large enough to reach the security threshold. When performing ρ repetitions, we can either simply repeat the exact protocol ρ times, at the cost of a almost ρ-fold increase in communication at every stage (preprocessing, proof size and communication between the verifiers; only the circuit wires need not be sent repeatedly), or we can slightly modify it to reduce the number of "full" repetitions by having the verifiers perform some σ Schwartz-Zippel checks for each repetition. Observe that the bound on the soundness error then becomes $\left(\frac{1}{\mathbb{F}} + \left(\frac{2\cdot(n_2+\sigma-1)}{|\mathbb{F}|-n_2}\right)^{\sigma}\right)^{\rho}$, since each Schwartz-Zippel check, performed with an independent element ζ, has an independent failure probability of $\frac{2\cdot(n_2+\sigma-1)}{|\mathbb{F}|-n_2}$. The only further modification to the protocol we need is for the verifiers to reveal ρ linear combinations and, to preserve the zero-knowledge property, for the prover to add ρ extra random points (t_i) to each polynomial, rather than just 1.

Theorem 3. *Let $\mathcal{H}$ be a random oracle, if $t < n/3$ then protocol $\Pi_{3\mathrm{t}}$ described in Fig. 6 secure implements the functionality $\mathcal{F}_{\mathrm{DV-ZK}}$ in the $(\mathcal{F}^{t,n}_{\mathrm{Prep}}, \mathcal{F}_{\mathrm{Rand}})$-hybrid model with $\Gamma_C = \Gamma_S = \Gamma_Z$ being the set of all subsets of verifiers of size $n - t$ or more.*

In the proof, we use the following lemma.

Lemma 6. *Let $a_\ell \cdot b_\ell \neq c_\ell$, for some $\ell \in [n_S]$, then the probability of passing the multiplication test if parties honestly perform the check is*

$$\frac{1}{|\mathbb{F}|} + \frac{2\cdot n_2}{|\mathbb{F}| - n_2}.$$

Proof. (of Lemma 6) Here we suppose that the r_j's are randomly chosen. Assuming that the check passes, then one of the following two conditions must hold:

1. The values $r_1, \ldots, r_1$ were sampled such that the multiplicative relation holds.
2. A value ζ was chosen such that $P(\zeta) = \sum_j A_j(\zeta) \cdot B_j(\zeta)$ while $P \neq T = \sum_j A_j \cdot B_j$.

Assuming that at least one triple is incorrect, the first relation holds with probability at most $\frac{1}{|\mathbb{F}|}$. In the second case, the polynomials on both sides are of degree $2 \cdot n_2$ and can have at most $2 \cdot n_2$ points in common. By the Schwartz-Zippel Lemma, the probability of choosing such a value of ζ is at most $\frac{2\cdot n_2}{|\mathbb{F}|-n_2}$. The overall soundness error is therefore at most $\frac{1}{|\mathbb{F}|} + \frac{2\cdot n_2}{|\mathbb{F}|-n_2}$. □

Proof. (of Theorem 3) The simulator $\mathcal{S}$ obtains as input from the environment the set $\mathcal{D}$ of corrupted parties and forwards $(\mathsf{Corrupt}, \mathcal{D})$ to the functionality. Throughout the execution, $\mathcal{S}$ simulates the random oracle $\mathcal{H}$ by answering every new query with a random value from the relevant set and maintaining a list of past queries to answer repeated queries consistently. As in the real protocol, the simulator uses a deterministic expansion function that for each seed defines a distinct random tape.

The simulation is very similar to that of Theorem 2. On input (Init) from $\mathcal{F}_{\mathrm{DV-ZK}}$, if $\mathcal{A}$ sends Abort, it forwards Abort to the functionality, otherwise forwards (ok). $\mathcal{S}$ sets up a copy of $\mathcal{F}_{\mathrm{Rand}}$ and emulates $\mathcal{F}^{t,n}_{\mathrm{Prep}}$ obtaining the values s_i and $s_i^{(j)}$, for $i \in [n_S + n_W + 2\cdot n_1 + n_2 + 2]$, held by corrupted parties. Since $\mathcal{V}_{\mathcal{D}} \in \Delta_Z$, it receives (Prove, x) from the functionality. If $\mathcal{P} \in \mathcal{H}$, then it randomly samples the shares of the proof for corrupted parties, and sets honest shares consistently, i.e., such that the multiplication values are correct and $o = 0$; otherwise it receives from $\mathcal{A}$ the proof. In this case, $\mathcal{S}$ reconstructs the input $\tilde{w}$ and forwards $(\mathsf{Prove}, x, \tilde{w})$ to the functionality. The simulator

$\mathcal{S}$ starts the simulation of the verification step, i.e. it evaluates the circuit honestly, for each gate computes the shares held by corrupted parties and sends the shares of the honest parties needed to run $\mathsf{RobustReconstruct}(\langle o \rangle_t, t)$. Receiving the shares of corrupted parties, it checks if those shares are the same as the ones computed by $\mathcal{S}$. We distinguish two different cases:

- If $\mathcal{P} \in \mathcal{H}$: The simulator $\mathcal{S}$ sets the flag accept. If the shares are consistent then $C_A = \emptyset$; else, if the shares are inconsistent, it identifies the cheating verifiers with incorrect shares and updates C_A with those parties.
- If $\mathcal{P} \notin \mathcal{H}$: if either $(x, w) \notin \mathcal{R}$ or some of the multiplication values given by the prover are incorrect, it sets the flag reject. If some of the shares are inconsistent, then $\mathcal{S}$ identifies cheaters and updates C_A.

After this, $\mathcal{S}$ emulates the Multiplications check. It samples random ζ, calling $\mathcal{F}_{\text{Rand}}$, and sends these values to $\mathcal{A}$. If at any time $\mathcal{F}_{\text{Rand}}$ sends (Abort, C_A), the simulator forwards (Abort, C_A) to the functionality. The simulator $\mathcal{S}$ recomputes the values r_j using their queries' list. It also sends to $\mathcal{A}$ the honest shares, necessary to run $\mathsf{RobustReconstruct}$ and receives from $\mathcal{A}$ the shares of dishonest verifiers. If some of these shares are incorrect, it updates C_A with the corresponding corruptions.

Finally, if the flag accept or reject is true, $\mathcal{S}$ sends $(\mathsf{Abort}, 1, C_A)$ to $\mathcal{F}_{\text{DV}-\text{ZK}}$, otherwise it sends $(\mathsf{Abort}, 0, C_A)$ to the functionality.

Indistinguishability. We now argue indistinguishability of the real and ideal executions to an environment, $\mathcal{Z}$. Recall that $\mathcal{Z}$ chooses the inputs of all parties. The view of $\mathcal{Z}$ in the real world then consists of these inputs, the messages received by the adversary and all the output values.

Indistinguishability of the proof follows from the privacy of Shamir's secret sharing scheme and from the fact that the input and the multiplication values are masked by the preprocessed values s_i, that are unknown to the adversary if the prover is honest. The messages received by the adversary in the multiplications check are randomized using random values sampled by the simulator, if $\mathcal{P}$ is honest, and unknown to $\mathcal{Z}$. From this and privacy of Shamir's sharings, simulation of these messages is perfect.

To argue indistinguishability of the output, we distinguish two cases as follows.

- If $\mathcal{P} \in \mathcal{H}$, the simulator always accepts the proof and outputs $(\mathsf{Abort}, 1, C_A)$ to the functionality, where the set $C_A = \emptyset$ if all the shares provided by $\mathcal{A}$ are correct and consistent. Irrespective of what the adversary does, $\mathsf{RobustReconstruct}$ always reconstructs the correct values, even with $\mathsf{flag} = (\mathsf{incorrect}, C)$, since $t < n/3$ and the sharings are correct since they are obtained by calling the preprocessing functionality. In the ideal execution, the simulator outputs the set of parties that provided incorrect shares, in the real one this same set is provided by $\mathsf{RobustReconstruct}$. Indeed, since the sharing is correct, it is possible to efficiently and correctly detect all the $t < n/3$ possible errors. Hence, in this case the simulation is perfect.
- If $\mathcal{P} \notin \mathcal{H}$, the simulator honestly evaluates the circuit with inputs extracted from the masked proof given by $\mathcal{A}$. Therefore, if $(x, \tilde{w}) \notin \mathcal{R}$ and the multiplication values that are part of the proof are correct, then $\mathcal{S}$ rejects the proof and the outputs of the two executions are identical. If $(x, \tilde{w}) \notin \mathcal{R}$ and the multiplication values are incorrect, then the simulator rejects the proof, whereas in the real execution the probability of acceptance is given by applying Lemma 6. In particular, denoting by q the number of oracle queries made by a malicious prover to $\mathcal{H}$, the probability that the first condition in the lemma holds is given by $q \cdot \frac{1}{|\mathbb{F}|}$ and the overall probability of passing the check is at most $q \cdot \frac{1}{|\mathbb{F}|} + \frac{2 \cdot n_2}{|\mathbb{F}| - n_2}$.

Protocol	Circuit	n	t	Field	Parameters	Number of preproc. element	Proof size (bytes)	Preprocessing Time (ms)	Prover Time (ms)	Verifier Time (ms)
$\Pi_{4\mathrm{t}}$	AES	5	1	$\mathbb{F}_{2^3}$	$\rho = 14$	7000	2496	1.17	2.12	4.25
$\Pi_{4\mathrm{t}}$	AES	6	1	$\mathbb{F}_{2^3}$	$\rho = 14$	7000	2496	1.36	2.12	3.92
$\Pi_{4\mathrm{t}}$	AES	7	1	$\mathbb{F}_{2^3}$	$\rho = 14$	7000	2496	1.43	2.22	4.43
$\Pi_{4\mathrm{t}}$	SHA-256	5	1	$\mathbb{F}_{2^3}$	$\rho = 14$	23000	8449	2.35	6.24	9.39
$\Pi_{4\mathrm{t}}$	SHA-256	6	1	$\mathbb{F}_{2^3}$	$\rho = 14$	23000	8449	2.63	6.51	9.76
$\Pi_{4\mathrm{t}}$	SHA-256	7	1	$\mathbb{F}_{2^3}$	$\rho = 14$	23000	8449	2.64	5.85	9.37
$\Pi_{3\mathrm{t}}$	AES	4	1	$\mathbb{F}_{2^{27}}$	$\rho = 3, \sigma = 2$	11000	26633	3.28	44.11	36.70
$\Pi_{3\mathrm{t}}$	AES	5	1	$\mathbb{F}_{2^{27}}$	$\rho = 3, \sigma = 2$	11000	26633	3.35	44.14	37.45
$\Pi_{3\mathrm{t}}$	AES	4	1	$\mathbb{F}_{2^{12}}$	$\rho = 7, \sigma = 3$	11000	16052	1.89	33.54	44.03
$\Pi_{3\mathrm{t}}$	AES	5	1	$\mathbb{F}_{2^{12}}$	$\rho = 7, \sigma = 3$	11000	16052	2.49	33.66	44.59
$\Pi_{3\mathrm{t}}$	AES	7	2	$\mathbb{F}_{2^{12}}$	$\rho = 7, \sigma = 3$	11000	16052	2.69	39.92	50.44
$\Pi_{3\mathrm{t}}$	SHA-256	4	1	$\mathbb{F}_{2^{27}}$	$\rho = 3, \sigma = 2$	32000	83553	9.06	229.93	63.59
$\Pi_{3\mathrm{t}}$	SHA-256	5	1	$\mathbb{F}_{2^{27}}$	$\rho = 3, \sigma = 2$	32000	83553	9.58	230.67	62.71
$\Pi_{3\mathrm{t}}$	SHA-256	4	1	$\mathbb{F}_{2^{12}}$	$\rho = 7, \sigma = 4$	32000	47802	4.46	159.72	75.65
$\Pi_{3\mathrm{t}}$	SHA-256	5	1	$\mathbb{F}_{2^{12}}$	$\rho = 7, \sigma = 4$	32000	47802	4.76	159.84	82.16
$\Pi_{3\mathrm{t}}$	SHA-256	7	2	$\mathbb{F}_{2^{12}}$	$\rho = 7, \sigma = 4$	32000	47802	6.31	167.76	89.82

Table 2. Experimental results for running the protocols in Fig. 5 and Fig. 6 on the AES and SHA-256 circuits

Finally, we conclude by observing that the set C_A of cheating verifiers is identical in both the executions and only consists of corrupted parties as this set in the protocol is given by running the Reed-Solomon reconstruction on a correct sharing with $t < n/3$.

□

7 Experiments

We present experimental validation of the efficiency of our protocols by presenting prover and verification times for proving knowledge of an AES-128 key corresponding to a public plaintext-ciphertext pair and a boolean circuit proving the knowledge of a preimage for the SHA-256 compression function. These functions where chosen as the boolean circuits for these are readily available, and well-studied. The AES-128 circuit has 6400 AND gates, while the SHA-256 circuit has 22573 AND gates. For both circuits and protocols we targeted a system tolerating a single corrupted verifier ($t = 1$) for a total of $n \in \{5, 6, 7\}$ verifiers for the case of protocol $\Pi_{4\mathrm{t}}$ and a total of $n \in \{4, 5, 7\}$ verifiers for the case of protocol $\Pi_{3\mathrm{t}}$. For the latter protocol, we also provide numbers for $t = 2$ corrupted parties for $n = 7$ parties in total.

Our experiments were run on a cluster of computers running Ubuntu 20.04.2 with a ping time of roughly 0.6 ms, and a total bandwith of 9.41Gbit/s per machine. The machines had either Intel i7-770K CPUs running at 4.2 GHz with 32 GB of RAM, or Intel i9-9900 CPUs running at 3.1 GHz with 128 GB of RAM. Each configuration was run a total of 200 times and the median was taken to obtain the presented running times.

Results. Our experimental results are presented in Table 2, and we can immediately see that $\Pi_{4\mathrm{t}}$ is an order of magnitude more efficient than $\Pi_{3\mathrm{t}}$. Given that a threshold of $t < n/4$ may be enough in a number of practical situations, one can see that $\Pi_{4\mathrm{t}}$ has runtimes which are efficient enough for practical deployment.

For protocol $\Pi_{4\mathrm{t}}$ from Fig. 5, targeting $t < n/4$, we selected the finite field $\mathbb{F}_{2^3}$ to accommodate the secret sharing. We performed $\rho = 14$ parallel repetitions of the protocol to boost the statistical security to 2^{-42}. When looking at the tradeoff between the field size of $\mathbb{F}_{2^k}$ and the number of repetitions ρ for this protocol, notice that the security level will always be $2^{-k\cdot\rho}$, regardless of how we distribute the load across the two parameters. Similarly, the communication cost among the verifiers does not depend on either ρ or k individually, but only on the product $\rho \cdot k$. Using a larger field size, however, does increase the proof size and the communication cost of the preprocessing phase, as those only depend on the field size, and not on ρ. Hence it should be preferred to use a smaller field with more parallel repetitions, rather than increasing the field size to target a security level for this protocol.

For protocol $\Pi_{3\mathrm{t}}$ in Fig. 6, targeting $t < n/3$, we again choose to aim for a security level of $\mathsf{sec} = 40$ and we let the maximum number of queries the prover can make to $\mathcal{H}$ be $q = 2^{40}$. As there is a trade-off between the field size, which influences the communication/proof size, and the number of full repetitions ρ to be performed, we provide data points for a small field $\mathbb{F}_{2^{12}}$ with $\rho = 7$, and for a larger field $\mathbb{F}_{2^{27}}$ with $\rho = 3$. In all cases, the minimal σ is chosen such that the security constraint $q \cdot \left(\frac{1}{|\mathbb{F}|} + \left(\frac{2\cdot(n_2+\sigma-1)}{|\mathbb{F}|-n_2}\right)^{\sigma}\right)^{\rho} \leq 2^{-\mathsf{sec}}$ is satisfied. We use a batch size $n_2 = 80$ for the AES circuit, and let $n_2 = 150$ for the SHA circuit. The number of parallel repetitions needed is always the minimal needed to have the failure probability bounded by $2^{-\mathsf{sec}}$, i.e. $\left\lceil \frac{\mathsf{sec}}{\log_2(|\mathbb{F}|)} \right\rceil$. From our experiments, the proof size is smaller for the smaller field, and the prover time is better, while the verifier time benefits more from having a larger field.

As the contribution to the proving time for $\Pi_{3\mathrm{t}}$ is the interpolation of the polynomials $P(x)$, which has complexity roughly in $O(|x| \cdot n_2)$, a tradeoff can be made to choose a smaller batch size n_2 at the cost of having a larger proof size due to n_1 growing.

Scaling to more verifiers. For both protocols, as we scale to a larger number of parties the dependence is essentially linear in terms of communication. The parameters ρ and σ do not depend on the number of parties at all. The round complexity of all protocols is independent of both n and t. The communication cost (in terms of amount of bytes sent by each verifier) in the verification protocol is $O(n)$ in the case of protocol $\Pi_{4\mathrm{t}}$, and is $O(n \cdot \sqrt{|x|})$ in the case of protocol $\Pi_{3\mathrm{t}}$; these both scale linearly with the number of verifiers, however, importantly they are sublinear in the total circuit size. So, for very large circuits our protocols should scale well with more parties. The threshold t has no effect on the round or total communication cost, it only increases the computational cost to perform a robust opening. To illustrate the scalability, we present experiments with increasing n up to 7 parties in Table 2. Note that we only see a small increase in verifier time for larger n, as the computation time dominates.

For our preprocessing protocol for any $t < n/2$, the dominant cost is each verifier sending $n_S/(n-t)$ shares to every other verifier, and to the prover. Therefore, if t is a constant fraction of n, the communication cost per verifier is linear in the circuit size but essentially independent of the total number of verifiers. For instance, with $t = n/3$ it is roughly $\frac{3}{2} \cdot n_S$ field elements, and for $t = n/4$ this becomes $\frac{4}{3} \cdot n_S$.

Comparison with other approaches. We have already made some comparisons with other systems in Section 1. Notice that [dDOS19, BdK+21, dSGOT21] report comparable prover and verification times for AES, however these papers use a more compact description of the AES circuit over $\mathbb{F}_{2^8}$ with S-boxes instead of AND gates. We could utilize the same approach, obtaining better runtimes.

However, our goal is different from the one in these papers as they specifically aim to obtain efficient post-quantum signature schemes based on AES, while we support general circuits, only using AES and SHA-256 as examples.

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) under contract HR001120C0085, by the FWO under an Odysseus project GOH9718N, by CyberSecurity Research Flanders with reference number VR20192203, by the Aarhus University Research Foundation, and by the Independent Research Fund Denmark under project number 0165-00107B.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

Protocol Π_{3t}

Let C be the circuit to be proved; the prover $\mathcal{P}$ is assumed to know an input witness w such that $C(w) = 0$. Let n_S denote the number of AND gates in the circuit and n_W the number of wires in the witness w. Write $n_S = n_1 \cdot n_2$.

Let CheckMult and OutputRec be two additional flags initially set to $\top$ and $\perp$ respectively; and $\mathcal{H}$ be a random oracle.

Init: Call $\mathcal{F}^{t,n}_{\text{Prep}}$ so that $\mathcal{P}$ obtains s_i and the verifiers $\mathcal{V}_1, \dots, \mathcal{V}_n$ obtain $\langle s_i \rangle_t$ for $i \in [n_S + n_W + 2n_1 + n_2 + 2]$, i.e. each $\mathcal{V}_j$ obtains the share $s_i^{(j)}, j \in [n]$.

Prove:

1. The prover computes the difference between the input values w_i and the preprocessed values s_i, i.e. $w_i - s_i, i \in [n_w]$.
2. $\mathcal{P}$ evaluates the circuit gate-by-gate:
   (a) For every linear gate, simply compute the resulting wire
   (b) For the j-th AND gate, compute the resulting wire $c_j = a_j \cdot b_j$ and compute $c_j - s_{j+n_W}$.
3. The prover samples $2 \cdot n_1$ random field elements t_i and computes the differences with preprocessed values $s_{n_W+n_S+i}, i \in [2n_1]$.
4. The prover computes n_1 random field elements r_j, seeded with $\mathcal{H}(\{w_i - s_i\}_i || \{c_i - s_{n_W+i}\}_i || \{t_i - s_{n_W+n_S+i}\}_i)$.
5. $\mathcal{P}$ computes the checking polynomials:
   (a) For $j \in [n_1]$, interpolate $A_j(x)$ from $r_j \cdot a_{j \cdot n_2 + k}$, $k \in [n_2]$, and $t_{2 \cdot j - 1}$
   (b) For $j \in [n_1]$, interpolate $B_j(x)$ from $b_{j \cdot n_2 + k}$ for $k \in [n_2]$ and $t_{2 \cdot j}$
   (c) Compute $P(x) = \sum_{j \in [n_1]} A_j(x) \cdot B_j(x)$ and $p_i = P(i + n_2 - 1)$ and $p_i - s_{n_W+n_S+2n_1+i}$, for $i \in [n_2 + 2]$.
6. The proof is given by the following values

$$\{w_i - s_i\}_{i \in [n_W]}, \{c_i - s_{n_W+i}\}_{i \in [n_S]}, \{t_i - s_{n_W+n_S+i}\}_{i \in [2n_1]}, \{p_i - s_{n_W+n_S+2 \cdot n_1+i}\}_{i \in [n_2+2]}.$$

Verify: The verifiers $\mathcal{V}_1, \dots, \mathcal{V}_n$ jointly check the circuit evaluation:

1. Evaluate the circuit within the Shamir secret sharing:
   (a) Shares of the input wires can be computed as $\langle s_i \rangle_t + (w_i - s_i)$ for $i \in [n_W]$.
   (b) Shares of the output wires from an AND gate can be computed as $\langle s_{j+n_W} \rangle_t + (c_j - s_{j+n_W})$ for $j \in [n_S]$.
   (c) Linear gates can be evaluated linearly over the shares
   (d) Let $\langle o \rangle_t$ be the sharing of the value of the output wire.
2. The verifiers call RobustReconstruct$(\langle o \rangle_t, t)$, to obtain (o, flag_o):
   If $o \neq 0$:
   - If $\mathsf{flag}_o = (\mathsf{correct}, \emptyset)$ then output Fail.
   - If $\mathsf{flag}_o = (\mathsf{incorrect}, C_o)$, then output the dishonest verifiers in C_o and Fail.

   Else, set OutputRec $= \top$. If $\mathsf{flag}_o = (\mathsf{incorrect}, C_o)$, identify the dishonest verifiers in C_o.
3. *Multiplications check*: The verifiers perform the following test.
   (a) Call $\zeta \leftarrow \mathcal{F}_{\text{Rand}}(\{\mathcal{V}_1, \dots, \mathcal{V}_n\}, \mathbb{F}_{2^k})$.
   (b) Update the shares of $s_{n_W+n_S+i}$ to obtain shares of t_i by computing $\langle t_i \rangle_t = \langle s_{n_W+n_S+i} \rangle_t - (t_i - s_{n_W+n_S+i})$, $i \in [2n_1]$.
   (c) Recompute the random values r_j, $j \in [n_1]$, produced by $\mathcal{P}$, by querying $\mathcal{H}$.
   (d) Interpolate polynomials $\langle A_j(x) \rangle_t$ and $\langle B_j(x) \rangle_t$, $j \in [n_1]$, using the wire values and $\langle t_i \rangle_t$, and polynomial $\langle P(x) \rangle_t$ using $\langle c_i \rangle_t$ and $\langle p_i \rangle_t$.
   (e) Run RobustReconstruct$(\langle P(\zeta) \rangle_t, t)$, RobustReconstruct$(\langle A_j(\zeta) \rangle_t, t)$ and RobustReconstruct$(\langle B_j(\zeta) \rangle_t, t)$, to obtain $(P(\zeta), \mathsf{flag}_P)$, $(A_j(\zeta), \mathsf{flag}_{A,j})$ and $(B_j(\zeta), \mathsf{flag}_{B_j})$ for $j \in [n_1]$.
   (f) Compute $T(\zeta) = \sum_{j \in [n_1]} A(\zeta) \cdot B(\zeta)$
   (g) If $P(\zeta) \neq T(\zeta)$ set CheckMult $= \perp$. Moreover,
      i. If all $\mathsf{flag}_{A,j} = \mathsf{flag}_{B,j} = \mathsf{flag}_P = (\mathsf{correct}, \emptyset)$ then output Fail
      ii. Otherwise, if for some j $\mathsf{flag}_{A,j} = (\mathsf{incorrect}, C_{A_j})$, $\mathsf{flag}_{B,j} = (\mathsf{incorrect}, C_{B_j})$ and/or $\mathsf{flag}_P = (\mathsf{incorrect}, C_P)$, identify dishonest verifiers and output Abort.
4. If both CheckMult $= \top$ and OutputRec $= \top$, accept the proof and identify possible dishonest verifiers identified in flag_o, flag_P, $\mathsf{flag}_{A,j}$ $\mathsf{flag}_{B,j}$.

Fig. 6. Protocol Π_{3t} for $t < n/3$

References

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2087–2104, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

BBC$^+$19. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 67–97, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

BBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

BCG$^+$13. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

BdK$^+$21. Carsten Baum, Cyprien de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 266–297, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany.

BGKW88. Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *20th Annual ACM Symposium on Theory of Computing*, pages 113–131, Chicago, IL, USA, May 2–4, 1988. ACM Press.

BMRS21. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.

CDG$^+$17. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1825–1842, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

dDOS19. Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*, volume 11959 of *Lecture Notes in Computer Science*, pages 669–692, Waterloo, ON, Canada, August 12–16, 2019. Springer, Heidelberg, Germany.

dSGOT21. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge mpcith-based arguments. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 3022–3036. ACM, 2021.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

HBHW16. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

HM97. Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In James E. Burns and Hagit Attiya, editors, *16th ACM Symposium Annual on Principles of Distributed Computing*, pages 25–34, Santa Barbara, CA, USA, August 21–24, 1997. Association for Computing Machinery.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30, San Diego, CA, USA, June 11–13, 2007. ACM Press.

JKO13. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 955–966, Berlin, Germany, November 4–8, 2013. ACM Press.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 525–537, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

LSTW21. Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time and post-quantum zero-knowledge snarks for r1cs. *Cryptology ePrint Archive*, 2021.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

WYKW21. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1074–1091. IEEE, 2021.

WZC$^+$18. Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 675–692, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2986–3001. ACM, 2021.