ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing

Ignacio Cascudo[1] and Bernardo David[2*]

[1] IMDEA Software Institute, Madrid, Spain, ignacio.cascudo@imdea.org
[2] IT University of Copenhagen, Copenhagen, Denmark, bernardo@bmdavid.com

Abstract. In this paper we present ALBATROSS, a family of multiparty randomness generation protocols with guaranteed output delivery and public verification that allows to trade off corruption tolerance for a much improved amortized computational complexity. Our basic stand alone protocol is based on publicly verifiable secret sharing (PVSS) and is secure under in the random oracle model under the decisional Diffie-Hellman (DDH) hardness assumption. We also address the important issue of constructing Universally Composable randomness beacons, showing two UC versions of Albatross: one based on simple UC NIZKs and another one based on novel efficient "designated verifier" homomorphic commitments. Interestingly this latter version can be instantiated from a global random oracle under the weaker Computational Diffie-Hellman (CDH) assumption. An execution of ALBATROSS with n parties, out of which up to $t = (1/2 - \epsilon) \cdot n$ are corrupt for a constant $\epsilon > 0$, generates $\Theta(n^2)$ uniformly random values, requiring in the worst case an amortized cost per party of $\Theta(\log n)$ exponentiations per random value. We significantly improve on the SCRAPE protocol (Cascudo and David, ACNS 17), which required $\Theta(n^2)$ exponentiations per party to generate one uniformly random value. This is mainly achieved via two techniques: first, the use of packed Shamir secret sharing for the PVSS; second, the use of linear t-resilient functions (computed via a Fast Fourier Transform-based algorithm) to improve the randomness extraction.

1 Introduction

Randomness is essential for constructing provably secure cryptographic primitives and protocols. While in many cases it is sufficient to assume that each party executing a cryptographic construction has access to a local trusted source of unbiased uniform randomness, many applications (*e.g.* electronic voting [1] and anonymous messaging [43,45]) require a randomness beacon [38] that can periodically provide fresh random values to all parties. Constructing such a randomness beacon without relying on a trusted third party requires a multiparty protocol that can be executed in such a way that all parties are convinced that an unbiased random value is obtained after the execution terminates, even if a fraction

*Work partially done while visiting IMDEA Software Institute. This work was supported by a grant from Concordium Foundation, DFF grant number 9040-00399B (TrA^{2}C) and Protocol Labs grant SLEDGE.

of these parties are corrupted. Moreover, in certain scenarios (*e.g.* in electronic voting [1]) it might be necessary to employ a publicly verifiable randomness beacon, which allows for third parties who did not participate in the beacon's execution to verify that indeed a given random value was successfully obtained after a certain execution. To raise the challenge of constructing such randomness beacons even more, there are classes of protocols that require a publicly verifiable randomness beacon with guaranteed output delivery, meaning that the protocol is guaranteed to terminate and output an unbiased random value no matter what actively corrupted parties do. A prominent class of protocols requiring publicly verifiable randomness beacons with guaranteed output delivery is that of Proof-of-Stake based blockchain consensus protocols [31,23], which are the main energy-efficient alternative to wasteful Proof-of-Work based blockchain consensus protocols [34,26].

Related Works: A number of randomness beacons aiming at being amenable to blockchain consensus applications have been proposed based on techniques such as Verifiable Delay Functions (VDF) [8], randomness extraction from data in the blockchain [2], Publicly Verifiable Secret Sharing [31,17,41] or Verifiable Random Functions [23,19]. However, most of these schemes do not guarantee either the generation of perfectly uniformly random values [2,23,19] or that a value will be generated regardless of adversarial behavior [41]. Those methods that do have those two guarantees suffer from high computational and communication complexity [31] or even higher computational complexity in order to improve communication complexity [8]. Another issue with VDF based approaches is that their security relies on very precise estimates of the average concrete complexity of certain computational tasks (*i.e.* how much time it takes an adversary to compute a VDF), which are hard to obtain for real world systems. While SCRAPE [17] does improve on [31], it can still be further improved, as is the goal of this work. Moreover, none of the protocols that guarantee generation of truly unbiased uniformly random values have any composability guarantees. This is a very important issue, since these protocols are not used in isolation but as building blocks of more complex systems and thus need composability.

Our Contributions: We present ALBATROSS, a family of multiparty randomness generation protocol with guaranteed output delivery and public verification, where parties generate $\Theta(n^2)$ independent and uniformly random elements in a group and where the computational complexity for each party in the worst case is of $\Theta(\log n)$ group exponentiations (the most computationally expensive operation in the protocol) per random element generated, as long as the number of corrupted parties is $t = n/2 - \Theta(n)$. Our contributions are summarized below:

- The first randomness beacon with $\Theta(\log n)$ group exponentiations per party.
- The first Universally Composable randomness beacon producing unbiased uniformly random values.
- The first randomness beacon based on the Computational Diffie-Hellman (CDH) assumption via novel "designated verifier" homomorphic commitments, which might be of independent interest.

Our basic stand alone protocol builds on SCRAPE [17], a protocol based on publicly verifiable secret sharing (PVSS). We depart from the variant of SCRAPE based on the Decisional Diffie-Hellman (DDH) assumption, which required $\Theta(n^2)$ group exponentiations per party to generate just one uniformly random element in the group, but tolerated any dishonest minority. Therefore, what we obtain is a trade-off of corruption tolerance in exchange for a much more efficient randomness generation, under the same assumptions (DDH hardness, RO model). We gain efficiency for ALBATROSS in the suboptimal corruption scenario by introducing two main techniques on top of SCRAPE, that in fact can be applied independently from each other: the first one is the use of packed Shamir secret sharing in the PVSS, and the second is the use of privacy amplification through t-resilient functions that allows to extract more uniform randomness from a vector of group elements from which the adversary may control some of the coordinates. Applying these techniques requires us to overcome significant obstacles (see below) but using them together allows ALBATROSS to achieve the complexity of $\Theta(\log n)$ exponentiations per party and random group element. Moreover, this complexity is worst case: the $\log n$ factor only appears if a large number of parties refuse to open the secrets they have committed to, thereby forcing the PVSS reconstruction on many secrets, and a less efficient output phase. Otherwise (if e.g. all parties act honestly) the amortized complexity is of $O(1)$ exponentiation per party and element generated.

Our Techniques: In order to create a uniformly random element in a group in a multiparty setting, a natural idea is to have every party select a random element of that group and then have the output be the group operation applied to all those elements. However, the last party in acting can see the choices of the other parties and change her mind about her input, so a natural solution is to have every party commit to their random choice first. Yet, the adversary can still wait until everyone else has opened their commitments and decide on whether they want to open or not based on the observed result, which clearly biases the output. In order to solve this, we can have parties commit to the secrets by using a publicly verifiable secret sharing scheme to secret-share them among the other parties as proposed in [31,17]. The idea is that public verifiability guarantees that the secret will be able to be opened even if the dealer refuses to reveal the secrets. The final randomness is constructed from all these opened secrets.

In the case of SCRAPE the PVSS consists in creating Shamir shares σ_i for a secret s in a finite field $\mathbb{Z}_q$, and publishing the encryption of σ_i under the public key pk_i of party i. More concretely, the encryption is $pk_i^{\sigma_i}$, and $pk_i = h^{sk_i}$ for h a generator of a DDH-hard group $\mathbb{G}_q$ of cardinality q; what party i can decrypt is not really the Shamir share σ_i, but rather h^{σ_i}. However these values are enough to reconstruct h^s which acts as a uniformly random choice in the group by the party who chose s. The final randomness is $\prod h^{s^a}$. Public verifiability of the secret sharing is achieved in SCRAPE by having the dealer commit to the shares independently via some other generator g of the group (i.e. they publish g^{σ_i}), proving that these commitments contain the same Shamir shares via discrete logarithm equality proofs, or DLEQs, and then having verifiers use a procedure

to check that the shares are indeed evaluations of a low-degree polynomial. In this paper we will use a different proof, but we remark that the latter technique, which we call $Local_{LDEI}$ test, will be of use in another part of our protocol (namely it is used to verify that h^s is correctly reconstructed).

In ALBATROSS we assume that the adversary corrupts at most t parties where $n - 2t = \ell = \Theta(n)$. The output of the protocol will be ℓ^2 elements of $\mathbb{G}_q$.

Larger Randomness via Packed Shamir Secret Sharing. In this sub-optimal corruption scenario, we can use *packed* Shamir secret sharing, which allows to secret-share a vector of ℓ elements from a field (rather than a single element). The key point is that every share is still one element of the field and therefore the sharing has the same computational cost ($\Theta(n)$ exponentiations) as using regular Shamir secret sharing. However, there is still a problem that we need to address: the complexity of the reconstruction of the secret vector from the shares increases by the same factor as the secret size (from $\Theta(n)$ to $\Theta(n^2)$ exponentiations). To mitigate this we use the following strategy: each secret vector will be reconstructed only by a random subset of c parties (independently of each other). Verifying that a reconstruction is correct only requires $\Theta(n)$ exponentiations, by using the aforementioned $Local_{LDEI}$. The point is that if we assign $c = \log n$, then with large probability there will be only at most a small constant number of secret tuples that were not correctly reconstructed by any of the $c(n)$ parties and therefore it does not add too much complexity for the parties to compute those. The final complexity of this phase is then $O(n^2 \log n)$ exponentiations for each party, in the worst case.

Larger Randomness via Resilient Functions. To simplfy, let us first assume that packed secret sharing has not been used. In that case, right before the output phase from SCRAPE, parties will know a value h^{s_a} for each of the parties P_a in the set $\mathcal{C}$ of parties that successfully PVSS'ed their secrets (to simplify, let us say $\mathcal{C} = \{P_1, P_2, \ldots, P_{|\mathcal{C}|}\}$), where h is a generator of a group of order q. In the original version of SCRAPE, parties then compute the final randomness as $\prod_{a=1}^{|\mathcal{C}|} h^{s_a}$, which is the same as $h^{\sum_{a=1}^{|\mathcal{C}|} s_a}$.

Instead, in ALBATROSS, we use a randomness extraction technique based on a linear t-resilient function, given by a matrix M, in such a way that the parties instead output a vector of random elements $(h^{r_1},...,h^{r_m})$ where $(r_1, ..., r_m) = M(s_1, \ldots, s_{|\mathcal{C}|})$. The resilient function has the property that the output vector is uniformly distributed as long as $|\mathcal{C}| - t$ inputs are uniformly distributed, even if the other t are completely controlled by the adversary. If in addition packed secret sharing has been used, one can simply use the same strategy for each of the ℓ coordinates of the secret vectors created by the parties. In this way we can create ℓ^2 independently distributed uniformly random elements of the group.

An obstacle to this randomness extraction strategy is that, in the presence of corrupted parties some of the inputs s_i may not be known if the dealers of these values have refused to open them, since PVSS reconstruction only allows to retrieve the values h^{s_i}. Then the computation of the resilient function needs to be done in the exponent which in principle appears to require either $O(n^3)$ exponentiations, or a distributed computation like in the PVSS reconstruction.

Fortunately, in this case the following idea allows to perform this computation much more efficiently: we choose M to be certain type of Vandermonde matrix so that applying M is evaluating a polynomial (with coefficients given by the s_i) on several n-th roots of unity. Then we adapt the Cooley-Tukey fast Fourier transform algorithm to work in the exponent of the group and compute the output with $n^2 \log n$ exponentiations, which in practice is almost as fast as the best-case scenario where the s_i are known. This gives the claim amortized complexity of $O(\log n)$ exponentiations per party and random element computed.

Additional Techniques to Decrease Complexity. We further reduce the complexity of the PVSS used in ALBATROSS, with an idea which can also be used in SCRAPE [17]. It concerns public verification that a published sharing is correct, i.e. that it is of the form $pk_i^{p(i)}$ for some polynomial of bounded degree, say at most k. Instead of the additional commitment to the shares used in [17], we use standard Σ-protocol ideas that allow to prove this type of statement, which turns out to improve the constants in the computational complexity. We call this type of proof a low degree exponent interpolation (LDEI) proof.

Universal Composability. We extend our basic stand alone protocol to obtain two versions that are secure in the Universal Composability (UC) framework [13], which is arguably one of the strongest security guarantees one can ask from a protocol. In particular, proving a protocol UC secure ensures that it can be used as a building block for more complex systems while retaining its security guarantees, which is essential for randomness beacons. We obtain the first UC-secure version of ALBATROSS by employing UC non-interactive zero knowledge proofs (NIZKs) for discrete logarithm relations, which can be realized at a reasonable overhead. The second version explores a new primitive that we introduce and construct called "designated verifier" homomorphic commitments, which allows a sender to open a commitment towards one specific receiver in such a way that this receiver can later prove to a third party that the opening revealed a certain message. Instead of using DDH based encryption schemes as before, we now have the parties commit to their shares using our new commitment scheme and rely on its homomorphic properties to perform the LDEI proofs that ensure share validity. Interestingly, this approach yields a protocol secure under the weaker CDH assumption in the random oracle model.

2 Preliminaries

$[n]$ denotes the set $\{1, 2, \ldots, n\}$ and $[m, n]$ denotes the set $\{m, m+1, \ldots, n\}$. We denote vectors with black font lowercase letters, i.e. $\mathbf{v}$. Given a vector $\mathbf{v} = (v_1, \ldots, v_n)$ and a subset $I \subseteq [n]$, we denote by $\mathbf{v}_I$ the vector of length $|I|$ with coordinates $v_i, i \in I$ in the same order they are in $\mathbf{v}$. Throughout the paper, q will be a prime number and $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ is a finite field of q elements. For a field $\mathbb{F}$, $\mathbb{F}^{m \times n}$ is the set of $m \times n$ matrices with coefficients in $\mathbb{F}$. Moreover, we denote by $\mathbb{F}[X]_{\leq m}$ the vector space of polynomials in $\mathbb{F}[X]$ with degree at most m. For a set $\mathcal{X}$, let $x \xleftarrow{\$} \mathcal{X}$ denote x chosen uniformly at random from $\mathcal{X}$; and for a distribution $\mathcal{Y}$, let $y \xleftarrow{\$} \mathcal{Y}$ denote y sampled according to the distribution $\mathcal{Y}$.

Polynomial Interpolation and Lagrange Basis. We recall a few well known facts regarding polynomial interpolation in fields.

Definition 1 (Lagrange basis). *Let $\mathbb{F}$ be a field, and $S = \{a_1, \ldots, a_r\} \subseteq \mathbb{F}$. A basis of $\mathbb{F}[X]_{\leq r-1}$, called the Lagrange basis for S, is given by $\{L_{a_i,S}(X) : i = 1, \ldots, r\}$ defined by*

$$L_{a_i,S}(X) = \prod_{a_j \in S \setminus \{a_i\}} \frac{X - a_j}{a_i - a_j}.$$

Lemma 1. *Let $\mathbb{F}$ be a field, and $S = \{a_1, \ldots, a_r\} \subseteq \mathbb{F}$. Then the map $\mathbb{F}[X]_{\leq r-1} \to \mathbb{F}^r$ given by $f \mapsto (f(a_1), \ldots, f(a_r))$ is a bijection, and the preimage of $(b_1, \ldots, b_r) \in \mathbb{F}^r$ is given by $f(X) = \sum_{i=1}^{r} b_i \cdot L_{a_i,S}(X)$.*

Packed Shamir Secret Sharing. From now on we work on the finite field $\mathbb{Z}_q$. Shamir secret sharing scheme [40] allows to share a secret $s \in \mathbb{Z}_q$ among a set of n parties (where $n < q$) so that for some specified $1 \leq t < n$, the secret can be reconstructed from any set of $t + 1$ shares via Lagrange interpolation ($t + 1$-reconstruction), while any t or less shares convey no information about it (t-privacy). In Shamir scheme each share is also in $\mathbb{Z}_q$ and therefore of the same size of the secret.

Packed Shamir secret sharing scheme ([7,25]) is a generalization that allows for sharing a vector in $\mathbb{Z}_q^\ell$ while each share is still one element of $\mathbb{Z}_q$. Standard Shamir is the case $\ell = 1$. Packing comes at the inevitable cost of sacrificing the threshold nature of Shamir's scheme, which is replaced by an (optimal) quasithreshold behavior, namely there is t-privacy and $t + \ell$ reconstruction.

The description of the sharing and reconstruction (from $t + \ell$ shares) algorithms can be found in Figure 1.

Remark 1. The points $0, -1, \ldots, -(\ell - 1)$ (for the secret) and $1, \ldots, n$ (for the shares) can be replaced by any set of $n + \ell$ pairwise distinct points. In this case the reconstruction coefficients should be changed accordingly. Choosing other evaluation points may be beneficial due to efficient algorithms for both computing the shares and the Lagrange coefficients [42]. In this paper, we will not focus on optimizing this aspect, and use the aforementioned points for simplicity.

Linear Codes. The Hamming weight of a vector $\mathbf{c} \in \mathbb{Z}_q^n$ is the number of nonzero coordinates of $\mathbf{c}$. An $[n, k, d]$ linear error correcting code C over $\mathbb{Z}_q$ is a vector subspace of $\mathbb{Z}_q^n$ of dimension k and minimum distance d, i.e., the smallest Hamming weight of a nonzero codeword in C is exactly d. A generator matrix is a matrix $M \in \mathbb{Z}_q^{k \times n}$ such that $C = \{\mathbf{m} \cdot M : \mathbf{m} \in \mathbb{Z}_q^k\}$.

Given n pairwise distinct points $x_1, \ldots, x_n$ in $\mathbb{Z}_q^n$, a Reed Solomon of length n and dimension k is defined as $= \{(f(x_1), \ldots, f(x_n)) : f \in \mathbb{Z}_q[X], \deg f < k\}$. It is well known that this is an $[n, k, n-k+1]$ linear code, and therefore achieves the largest possible minimum distance for a code of that length and dimension. These codes are called MDS (maximum distance separable).

The dual code of a code C, denoted $C^\perp$, is the vector space consisting of all vectors $\mathbf{c}^\perp \in \mathbb{Z}_q^n$ such that $\langle \mathbf{c}, \mathbf{c}^\perp \rangle = 0$ for all $\mathbf{c} \in C$ where $\langle \cdot, \cdot \rangle$ denotes

Packed Shamir secret sharing

Packed Shamir secret sharing over $\mathbb{Z}_q$ for ℓ secrets with n parties, t-privacy and $t+\ell$-reconstruction. We require $n+\ell \leq q$, $1 \leq t$, $t+\ell \leq n$.

Sharing algorithm.

On input $(s_0, s_1, \ldots, s_{\ell-1}) \in \mathbb{Z}_q^\ell$:

- The dealer chooses a polynomial uniformly at random in the affine space

$$\{f \in \mathbb{Z}_q[X]_{\leq t+\ell-1}, f(0) = s_0, f(-1) = s_1, \ldots, f(-(\ell-1)) = s_{\ell-1}\}.$$

- For $i = 1, \ldots, n$, the dealer sends $f(i)$ to the i-th party.

Reconstruction algorithm.

On input the shares $\sigma_i = f(i), i \in \mathcal{Q}$ for a set of parties $\mathcal{Q} \subseteq [n]$, with $|\mathcal{Q}| = t+\ell$.

- For $m = 0, \ldots, \ell-1$, parties compute

$$s_m = \sum_{i \in \mathcal{Q}} \sigma_i L_{i,\mathcal{Q}}(-m) = \sum_{i \in \mathcal{Q}} \sigma_i \prod_{j \in \mathcal{Q}, j \neq i} \frac{-m-j}{i-j}$$

- Output $(s_0, s_1, \ldots, s_{\ell-1})$

Fig. 1. Packed Shamir Secret Sharing (Sharing Algorithm)

the standard inner product. For the Reed-Solomon code above, its dual is the following so-called generalized Reed-Solomon code

$$C^\perp = \{(u_1 \cdot f_*(x_1), \ldots, u_n \cdot f_*(x_n)) : g \in \mathbb{Z}_q[X], \deg f_* < n-k\}$$

where $u_1, ..., u_n$ are fixed elements of $\mathbb{Z}_q^n$, namely $u_i = \prod_{j=1, j\neq i}^n (x_i - x_j)^{-1}$.

Linear Perfect Resilient Functions. Our optimizations make use of randomness extractors which are linear over $\mathbb{Z}_q$ and hence given by a matrix $M \in \mathbb{Z}_q^{u \times r}$ satisfying the following property: the knowledge of any t coordinates of the input gives no information about the output (as long as the other $r-t$ coordinates are chosen uniformly at random). This notion is known as linear perfect t-resilient function [20].

Definition 2. *A $\mathbb{Z}_q$-linear (perfect) t-resilient function (t-RF for short) is a linear function $\mathbb{Z}_q^r \to \mathbb{Z}_q^u$ given by $\mathbf{x} \mapsto M \cdot \mathbf{x}$ such that for any $I \subseteq [r]$ of size t, and any $\mathbf{a}_I = (a_j)_{j \in I} \in \mathbb{Z}_q^t$, the distribution of $M \cdot \mathbf{x}$ conditioned to $\mathbf{x}_I = \mathbf{a}_I$ and to $\mathbf{x}_{[r]\setminus I}$ being uniformly random in $\mathbb{Z}_q^{r-t}$, is uniform in $\mathbb{Z}_q^u$.*

Note that such a function can only exist if $u \leq r-t$. We have the following characterization in terms of linear codes.

Theorem 1. *[20] An $u \times r$ matrix M induces a linear t-RF if and only if M is a generator matrix for an $[r, u, t+1]$-linear code over $\mathbb{Z}_q$.*

Remark 2. Remember that with our notation for linear codes, the generator matrix acts on the right for encoding a message, i.e. $\mathbf{m} \mapsto \mathbf{m} \cdot M$. In other words

the encoding function for the linear code and the corresponding resilient function given by the generator matrix as in Theorem 1 are "transpose from each other".

A t-RF for the optimal case $u = r - t$ is given by any generator matrix of an $[r, r-t, t+1]$ MDS code, for example a matrix M with $M_{ij} = a_j^{i-1}$ for $i \in [r-t]$, $j \in [r]$, where all a_j's are distinct, which generates a Reed-Solomon code. It will be advantageous for us to fix an element $\omega \in \mathbb{Z}_q^*$ of order at least $r - t$ and set $a_j = \omega^{j-1}$, that is we will use the Vandermonde matrix $M = M(\omega, r-t, r)$ where

$$M_{ij} = \omega^{(i-1)(j-1)}, i \in [r-t], j \in [r]$$

Then $M \cdot \mathbf{x} = (f(1), f(\omega), \cdots, f(\omega^{r-t-1}))$ where $f(X) := x_0 + x_1 X + x_2 X^2 + \cdots + x_{r-1} X^{r-1}$, and we can use the Fast Fourier transform to compute $M \cdot \mathbf{x}$ very efficiently, as we explain later.

Remark 3. [22] An even more efficiently (at least aymptotically) computable family of resilient functions, albeit not explicit and with worse parameters, is given as follows: choosing random $0-1$ matrices over $\mathbb{Z}_q$ of certain dimension $b \cdot r$ for some small constant b will give, with high probability, a $[r, b \cdot r, c \cdot r]$-linear code over $\mathbb{Z}_q$ for some small c, and therefore a $c \cdot r$-RF functions where the image is of size $b \cdot r$.

Nevertheless, we will use the resilient function based on the Vandermonde matrix above, because of its optimal parameters and the fact that the FFT-based computation is very efficient in practice.

3 Basic Algorithms and Protocols

In this section we introduce some algorithms and subprotocols which we will need in several parts of our protocols, and which are relatively straight-forward modifications of known techniques.

3.1 Proof of Discrete Logarithm Equality.

We will need a zero-knowledge proof that given $g_1, ..., g_m$ and $x_1, ..., x_m$ the discrete logarithms of every x_i with base g_i are equal. That is $x_i = g_i^\alpha$ for all $i = 1, \ldots, m$ for some $\alpha \in \mathbb{Z}_q$ (the same α for all i). Looking ahead, these proofs will be used by parties in the PVSS to ensure they have decrypted shares correctly. A sigma-protocol performing DLEQ proofs for $m = 2$ was given in [18]. We can easily adapt that protocol to general m as follows:

1. The prover samples $w \leftarrow \mathbb{Z}_q$ and, for all $i = 1, \ldots, m$, computes $a_i = g_i^w$ and sends a_i to the verifier.
2. The verifier sends a challenge $e \leftarrow \mathbb{Z}_q$ to the prover.
3. The prover sends a response $z = w - \alpha e$ to the verifier.
4. The verifier accepts if $a_i = g_i^z x_i^e$ for all $i = 1, \ldots, m$.

We transform this proof into a non-interactive zero-knowledge proof of knowledge of α in the random oracle model via the Fiat-Shamir heuristic [24,37]:

- The prover computes $e = H(g_1, \ldots, g_m, x_1, \ldots, x_m, a_1, \ldots, a_m)$, for $H(\cdot)$ a random oracle (that will be instantiated by a cryptographic hash function) and z as above. The proof is $(a_1, \ldots, a_m, e, z)$.
- The verifier checks that $e = H(g_1, \ldots, g_m, x_1, \ldots, x_m, a_1, \ldots, a_m)$ and that $a_i = g_i^z x_i^e$ for all i.

This proof requires m exponentiations for the prover and $2m$ for the verifier.

3.2 Proofs and Checks of Low-Degree Exponent Interpolation.

We consider the following statement: given generators $g_1, g_2, \ldots, g_m$ of a cyclic group $\mathbb{G}_q$ of prime order q, pairwise distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_m$ in $\mathbb{Z}_q$ and an integer $1 \leq k < m$, known by prover and verifier, the claim is that a tuple $(x_1, x_2, \ldots, x_m) \in \mathbb{G}_q^m$ is of the form $(g_1^{p(\alpha_1)}, g_2^{p(\alpha_2)}, \ldots, g_m^{p(\alpha_m)})$ for a polynomial $p(X)$ in $\mathbb{Z}_q[X]_{\leq k}$. We will encounter this statement in two different versions:

- In the first situation, we need a zero-knowledge proof of knowledge of $p(X)$ by the prover. This type of proof will be used for a dealer in the publicly verifiable secret sharing scheme to prove correctness of sharing. We call this proof $LDEI((g_i)_{i\in[m]}, (\alpha_i)_{i\in[m]}, k, (x_i)_{i\in[m]})$.
- In the second situation, we have no prover, but on the other hand we have $g_1 = g_2 = \cdots = g_m$. In that case we will use a locally computable check from [17]: indeed, verifiers can check by themselves that the statement is correct with high probability. This type of check will be used to verify correctness of reconstruction of a (packed) secret efficiently. We call such check $Local_{LDEI}((\alpha_i)_{i\in[m]}, k, (x_i)_{i\in[m]}))$. [3]

In [17], the first type of proof was constructed by using a DLEQ proof of knowledge of common exponent to reduce that statement to one of the second type and then using the local check we just mentioned. However, this is unnecessarily expensive both in terms of communication and computation. Indeed, a simpler Σ-protocol for that problem is given in Figure 2.

Proposition 1. *Protocol LDEI in Figure 2 is an honest-verifier zero-knowledge proof of knowledge for the given statement.*

Proof. If the prover is honest, then clearly z is of degree $\leq k$ and $x_i^e \cdot a_i = g_i^{e \cdot p(\alpha_i)} \cdot g_i^{r(\alpha_i)} = g_i^{z(\alpha_i)}$. To prove soundness, given the first message $(a_1, a_2, \ldots, a_m)$, if the prover can provide z, z' that pass the test for two different challenges e, e', then $x_i^{e-e'} = g_i^{(z-z')(\alpha_i)}$ for all i. Therefore $x_i = g_i^{p(\alpha_i)}$, where $p(X) =$

[3] This type of statement is independent of the generator g_1 of the group we choose: it is true for a given generator if and only if it is true for all of them.

Protocol *LDEI* (Zero Knowledge Proof of Low-Degree Exponent Interpolation)

Public parameters: prime q, cyclic group $\mathbb{G}_q$ of prime order q, $g_1, ..., g_m$ generators of $\mathbb{G}_q$, $\alpha_1, \alpha_2, \ldots, \alpha_m$ pairwise distinct elements in $\mathbb{Z}_q$, integer $1 \leq k < m$.
Statement: $(x_1, x_2, \ldots, x_m) \in \left\{ \left(g_1^{p(\alpha_1)}, g_2^{p(\alpha_2)}, \ldots, g_m^{p(\alpha_m)} \right) : p \in \mathbb{Z}_q[X], \deg p \leq k \right\}$ and the prover knows p.

Protocol:
- Sender chooses $r(X) \in \mathbb{Z}_q[X]_{\leq k}$ uniformly at random and sends $a_i = g_i^{r(\alpha_i)}$ for all $i = 1, \ldots, m$ to the verifier.
- Verifier chooses $e \in \mathbb{Z}_q$ uniformly at random.
- Sender sends $z(X) = e \cdot p(X) + r(X)$ to the verifier
- Verifier checks that $z(X) \in \mathbb{Z}_q[X]_{\leq k}$ and that $x_i^e \cdot a_i = g_i^{z(\alpha_i)}$ holds for all $i = 1, \ldots, m$.

Fig. 2. Protocol *LDEI* Zero Knowledge Proof of Low-Degree Exponent Interpolation

$(e - e')^{-1}(z(X) - z'(X))$ is of degree at most k. To prove honest verifier zero-knowledge, we construct the following simulator: it samples $z(X)$ and e respectively uniformly at random in $\mathbb{Z}_q[X]_{\leq k}$ and $\mathbb{Z}_q$, and it generates $a_i = g_i^{z(\alpha_i)}/x_i^e$ for all i. Then $a_i = g_i^{r(\alpha_i)}$, for $r(X) = z(X) - e \cdot p(X)$ and hence clearly r is uniformly random in $\mathbb{Z}_q[X]_{\leq k}$, so the distribution is as in the real proof.

Applying Fiat-Shamir heuristic we transform this into a non-interactive proof:

- The sender chooses $r \in \mathbb{Z}_q[X]_{\leq k}$ uniformly at random and computes $a_i = g_i^{r(\alpha_i)}$ for all $i = 1, \ldots, m$, $e = H(x_1, x_2, \ldots, x_m, a_1, a_2, \ldots, a_m)$ and $z = e \cdot p + r$. The proof is then $(a_1, a_2, \ldots, a_m, e, z)$.
- The verifier checks that $z \in \mathbb{Z}_q[X]_{\leq k}$, that $x_i^e \cdot a_i = g_i^{z(\alpha_i)}$ holds for all $i = 1, \ldots, m$ and that $e = H(x_1, x_2, \ldots, x_m, a_1, a_2, \ldots, a_m)$.

Now we consider the second type of situation mentioned above. The local check is given in Figure 3.

Proposition 2. *The local test $Local_{LDEI}$ in Figure 3 always accepts if the statement is true and rejects with probability at least $1 - 1/q$ if the statement is false.*

Correctness is based on the fact that the vector $(u_1 p_*(\alpha_1), \ldots, u_n p_*(\alpha_m))$ is in the dual code $C^\perp$ of the Reed Solomon code C given by the vectors $(p(\alpha_1), \ldots, p(\alpha_m))$ with $\deg p \leq k$, hence if the exponents of the x_i's (in base g) indeed form a codeword in C, the verifier is computing the inner product of two orthogonal vectors in the exponent. Soundness follows from the fact that, if the vector is not a codeword in C, then a uniformly random element in $C^\perp$ will only be orthogonal to that vector of exponents with probability less than $1/q$. See [17, Lemma 1] for more information about this claim.

Algorithm $Local_{LDEI}$ to Verify Low-Degree Exponent Interpolation

Public parameters: prime q, cyclic group $\mathbb{G}_q$ of prime order q, integer m.
Input: pairwise distinct elements $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ in $\mathbb{Z}_q$, integer $1 \leq k < m$, tuple $(x_1, x_2, \ldots, x_m) \in \mathbb{G}_q$, a group generator g.
Statement: $(x_1, x_2, \ldots, x_m) \in \left\{ \left(g^{p(\alpha_1)}, g^{p(\alpha_2)}, \ldots, g^{p(\alpha_m)} \right) : p \in \mathbb{Z}_q[X], \deg p \leq k \right\}$.
Algorithm:

- Verifier defines $u_i = 1/\prod_{\ell \neq i}(\alpha_i - \alpha_\ell)$ for all $i = 1, \ldots, m$.
- Verifier chooses a polynomial p_* uniformly at random in $\mathbb{Z}_q[X]_{\leq m-k-2} \setminus \{0\}$ and computes $v_i = u_i \cdot p_*(\alpha_i)$ for all i.
- Verifier checks that $\prod_{i=1}^{m} x_i^{v_i} = 1$ and accepts if and only if that is the case.

Fig. 3. Algorithm $Local_{LDEI}$ to Verify Low-Degree Exponent Interpolation

3.3 Applying Resilient Functions "in the Exponent"

In our protocol we will need to apply resilient functions in the following way. Let $h_1, \ldots, h_r$ be public elements of $\mathbb{G}_q$, chosen by different parties, so that $h_i = h^{x_i}$ (for some certain public generator h of the group) and x_i is only known to the party that has chosen it. Our goal is to extract $(\widehat{h}_1, \ldots, \widehat{h}_u) \in \mathbb{G}_q^u$ which is uniformly random in the view of an adversary who has control over up to t of the initial elements x_i. In order to do that, we take a t-resilient function from $\mathbb{Z}_q^r$ to $\mathbb{Z}_q^u$ given by a matrix M and apply it to the exponents, i.e., we define $\widehat{h}_i = h^{y_i}$ where $\mathbf{x} \mapsto \mathbf{y} = M \cdot \mathbf{x}$; this satisfies the desired properties. Because the resilient function is linear, the values $\widehat{h}_i$ can be computed from the h_i by group operations, without needing the exponents x_i. We define the following notation.

Definition 3. *As above, let $\mathbb{G}_q$ be a group of order q in multiplicative notation. Given a matrix $M = (M_{ij})$ in $\mathbb{Z}_q^{u \times r}$ and a vector $\mathbf{h} = (h_1, h_2, \ldots, h_r) \in \mathbb{G}_q^r$, we define $\widehat{\mathbf{h}} = M \diamond \mathbf{h} \in \mathbb{G}_q^u$, as $\widehat{\mathbf{h}} = (\widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_r)$, where $\widehat{h}_i = \prod_{k=1}^{u} h_k^{M_{ik}}$.*

Remark 4. Given a generator h of $\mathbb{G}_q$, if we write $\mathbf{h} = (h^{x_1}, h^{x_2}, \ldots, h^{x_r})$, $\mathbf{x} = (x_1, x_2, \ldots, x_r)$, then $M \diamond \mathbf{h} = (h^{y_1}, h^{y_2}, \ldots, h^{y_r})$ where $(y_1, y_2, \ldots, y_r) = M \cdot \mathbf{x}$.

Now let $M = M(\omega, r-t, r)$ as in Section 2. In order to minimize the number of exponentiations that we need to compute $M \diamond \mathbf{h}$ recall first that $M \cdot \mathbf{x} = (f(1), f(\omega), \ldots, f(\omega^{r-t-1}))$, where f is the polynomial with coefficients $f_i = x_{i+1}$, for $i \in [0, r-1]$. Assuming there exists $n > r - t - 1$ a power of 2 that divides $q - 1$, we can choose ω to be a n-th root of unity for n and use the well known Cooley-Tukey recursive algorithm [21] for computing the Fast Fourier Transform. The algorithm in fact evaluates a polynomial of degree up to $n - 1$ on all powers of ω up to ω^{n-1} with $O(n \log n)$ multiplications. We can just set $f_j = 0$ for $j \geq r$, and ignore the evaluations in ω^i, for $i \geq r-t$). For completeness, we include the algorithm as presented in [44] in Figure 15, Appendix A. In our situation the x_i's are not known; we use the fact that in the Cooley-Tukey

algorithm all operations on the x_i are linear, so we can operate on the values $h_i = h^{x_i}$ instead. The resulting algorithm is then given in Figure 4 (since we denoted $f_i = x_{i+1}$, then $h_i = h^{f_{i-1}}$).

"Cooley-Tukey FFT in the exponent" algorithm $FFTE$

Parameters: A large prime q, and a group $\mathbb{G}_q$ of cardinality q.
Input: An integer $n = 2^k$ dividing $q-1$, a tuple $\mathbf{h} = (h_1, h_2, \ldots, h_n) \in \mathbb{G}_q^n$, and an n-th root of unity $\omega \in \mathbb{Z}_q$.
Output: The tuple $\widehat{\mathbf{h}} = (\widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_n) = M' \diamond \mathbf{h} \in \mathbb{G}_q^n$, where $M' \in \mathbb{Z}_q^{n \times n}$ is given by $M'_{ij} = \omega^{(i-1)(j-1)}$ for $i, j \in [n]$.

If $n = 1$, return h_1.
Else:
- For $j = 1, \ldots, n/2$, compute $v_j = h_j \cdot h_{j+n/2}$, $v_j^* = (h_j \cdot (h_{j+n/2})^{-1})^{\omega^{j-1}}$. Set $\mathbf{v} = (v_1, v_2, \ldots, v_{n/2})$, $\mathbf{v}^* = (v_1^*, v_2^*, \ldots, v_{n/2}^*)$.
- Apply the algorithm recursively to $(n/2, \mathbf{v}, \omega^2)$ and on $(n/2, \mathbf{v}^*, \omega^2)$ obtaining outputs $\widehat{\mathbf{v}} = (\widehat{v}_1, \widehat{v}_2, \ldots, \widehat{v}_{n/2})$ and $\widehat{\mathbf{v}^*} = (\widehat{v^*}_1, \widehat{v^*}_2, \ldots, \widehat{v^*}_{n/2})$ respectively.
- Return $(\widehat{v}_1, \widehat{v^*}_1, \widehat{v}_2, \widehat{v^*}_2, \ldots, \widehat{v}_{n/2}, \widehat{v^*}_{n/2})$.

Fig. 4. Algorithm $FFTE$ (Cooley-Tukey FFT in the exponent)

At every recursion level of the algorithm, it needs to compute in total n exponentiations, and therefore the total number of exponentiations in $\mathbb{G}_q$ is $n \log_2 n$. In fact, half of these are inversions, which are typically faster.

4 ALBATROSS Protocols

We will now present our main protocols for multiparty randomness generation. We assume n participants, at most $t < (n-1)/2$ of which can be corrupted by some active static adversary. We define then $\ell = n - 2t > 0$. Note that $n - t = t + \ell$, so we use these two quantities interchangeably. For asymptotics, we consider that both t and ℓ are $\Theta(n)$, in particular $t = \tau \cdot n$ for some $0 < \tau < 1/2$. The n participants have access to a public ledger, where they can publish information that can be seen by the other parties and external verifiers.

Our protocols take place in a group $\mathbb{G}_q$ of prime cardinality q, where we assume that the Decisional Diffie-Hellman problem is hard. Furthermore, in order to use the FFTE algorithm we require that $\mathbb{G}_q$ has large 2-adicity, i.e., that $q-1$ is divisible by a large power of two 2^u. Concretely we need $2^u > n - t$. DDH-hard elliptic curve groups with large 2-adicity are known, for example both the Tweedledee and Tweedledum curves from [10] satisfy this property for $u = 33$, which is more than enough for any practical application.

4.1 A PVSS Based on Packed Shamir Secret Sharing

As a first step, we show a generalization of a PVSS from [17], where we use packed Shamir secret sharing in order to share several secrets at essentially the same cost for the sharing and public verification phases. In addition, correctness of the shares is instead verified using the $LDEI$ proof. This is different than in [17] where the dealer needed to commit to the shares using a different generator of the group, and correctness of the sharing was proved using a combination of DLEQ proofs and the $Local_{LDEI}$ check, which is less efficient. In Figure 5, we present the share distribution and verification of the correctness of the shares of the new PVSS. We discuss the reconstruction of the secret later.

Protocol π_{PPVSS}

Let h be a generator of a group $\mathbb{G}_q$ of order q. Let $H(\cdot)$ be a random oracle. Protocol π_{PPVSS} is run between n parties $P_1, \ldots, P_n$, a dealer D and an external verifier V (in fact any number of external verifiers) who have access to a public ledger where they can post information for later verification.

1. **Setup:** Party P_i generates a secret key $sk_i \leftarrow \mathbb{Z}_q$, a public key $pk_i = h^{sk_i}$ and registers the public key pk_i by posting it to the public ledger, for $1 \leq i \leq n$.
2. **Distribution:** The dealer D samples a polynomial $p(X) \leftarrow \mathbb{Z}_q[X]_{\leq t+\ell-1}$ and sets $s_0 = p(0), s_1 = p(-1), \ldots, s_{\ell-1} = p(-(\ell-1))$. The secrets are defined to be $S_0 = h^{s_0}, S_1 = h^{s_1}, \ldots, S_{\ell-1} = h^{s_{\ell-1}}$. D computes Shamir shares $\sigma_i = p(i)$ for $1 \leq i \leq n$. D encrypts the shares as $\hat{\sigma}_i = pk_i^{\sigma_i}$ and publishes $(\hat{\sigma}_1, \ldots, \hat{\sigma}_n)$ in the public ledger along with the proof $LDEI$ that $\hat{\sigma}_i = pk_i^{p(i)}$ for some p of degree at most $t + \ell - 1$.
3. **Verification:** The verifier checks the proof $LDEI$.

Fig. 5. Protocol π_{PPVSS}

Under the DDH assumption, π_{PPVSS} satisfies the property of IND1-secrecy as defined in [17] (adapted from [39,30]), which requires that given t shares and a vector $\mathbf{x}' = (s'_0, s'_1, \ldots, s'_{\ell-1})$, the adversary cannot tell whether $\mathbf{x}'$ is the actual vector of secrets.

Definition 4. ***Indistinguishability of secrets (IND1-secrecy)*** *We say that the PVSS is IND1-secret if for any polynomial time adversary $\mathcal{A}_{Priv}$ corrupting at most $t-1$ parties, $\mathcal{A}_{Priv}$ has negligible advantage in the following game played against a challenger.*

1. *The challenger runs the Setup phase of the PVSS as the dealer and sends all public information to $\mathcal{A}_{Priv}$. Moreover, it creates secret and public keys for all honest parties, and sends the corresponding public keys to $\mathcal{A}_{Priv}$.*
2. *$\mathcal{A}_{Priv}$ creates secret keys for the corrupted parties and sends the corresponding public keys to the challenger.*

3. *The challenger chooses values* $\mathbf{x}_0$ *and* $\mathbf{x}_1$ *at random in the space of secrets. Furthermore it chooses* $b \leftarrow \{0,1\}$ *uniformly at random. It runs the Distribution phase of the protocol with* x_0 *as secret. It sends* $\mathcal{A}_{Priv}$ *all public information generated in that phase, together with* $\mathbf{x}_b$.
4. $\mathcal{A}_{Priv}$ *outputs a guess* $b' \in \{0,1\}$.

The advantage of $\mathcal{A}_{Priv}$ *is defined as* $|\Pr[b = b'] - 1/2|$.

Proposition 3. *Protocol* π_{PPVSS} *is IND1-secret under the DDH assumption.*

We prove this proposition in Appendix B. The proof follows from similar techniques as in the security analysis of the PVSS in SCRAPE [17] and shows IND1-secrecy based on the ℓ-DDH hardness assumption, which claims that given $(g, g^{\alpha}, g^{\beta_0}, g^{\beta_1}, \cdots, g^{\beta_{\ell-1}}, g^{\gamma_0}, g^{\gamma_1}, \cdots, g^{\gamma_{\ell-1}})$ where the γ_i either have all been sampled at random from $\mathbb{Z}_q$ or are equal to $\alpha \cdot \beta_i$, it is hard to distinguish both situations. However, when ℓ is polynomial in the security parameter (as is the case here) ℓ-DDH is equivalent to DDH, see [35].

We now discuss how to reconstruct secrets in π_{PPVSS}. Rather than giving one protocol, in Figure 6 we present a number of subprotocols that can be combined in order to reconstruct a secret. The reason is to have some flexibility about which parties will execute the reconstruction algorithm and which ones will verify the reconstruction in the final randomness generation protocol.

In the share decryption protocol party P_i, using secret key sk_i, decrypts the share $\hat{\sigma}_i$ and publishes the obtained value h^{σ_i}. Moreover P_i posts a DLEQ proof to guarantee correctness of the share decryption; if several secret tuples need to be reconstructed, this will be done by a batch DLEQ proof.

Once $n - t$ values h^{σ_i} have been correctly decrypted (by a set of parties $\mathcal{Q}$), any party can compute the ℓ secret values $S_j = h^{s_j}$ using the reconstruction algorithm $Rec_{\mathcal{Q}}$, which boils down to applying Lagrange interpolation in the exponent. Note that since Lagrange interpolation is a linear operation, the exponents σ_i do not need to be known, one can operate on the values h^{σ_i} instead.

However, the computational complexity of this algorithm is high ($O(n^2)$ exponentiations) so we introduce the reconstruction verification algorithm $RecVer_{\mathcal{Q}}$ which allows any party to check whether a claimed reconstruction is correct at a reduced complexity ($O(n)$ exponentiations). $RecVer_{\mathcal{Q}}$ uses the local test $Local_{LDEI}$ that was presented in Figure 3.

In Table 2 (Appendix C), we collect the exact number of exponentiations required by each party in the PVSS. Here we remark that the more expensive computation is reconstruction of a secret which requires $O(n^2)$ exponentiations.

4.2 Scheduling of non-private computations

In ALBATROSS, parties may need to carry out a number of computations of the form $M \diamond \mathbf{h}$, where $M \in \mathbb{Z}_q^{r \times m}$, $\mathbf{h} \in \mathbb{G}_q^m$ for some $r, m = O(n)$. This occurs if parties decide not to reveal their PVSSed secrets, and it happens at two moments of the computation: when reconstructing the secrets from the PVSS and when applying the resilient function at the output phase of the protocol.

Reconstruction protocol and algorithms in π_{PPVSS}

Protocols used in the reconstruction of secrets in PVSS π_{PPVSS} from Figure 5. Same conditions and notations as there.

- **Share decryption (for P_i):** On input $\hat{\sigma}_i, pk_i$, decrypt share $\tilde{\sigma}_i = \hat{\sigma}_i^{\frac{1}{sk_i}} = h^{\sigma_i}$ and publish it in the ledger together with $PROOF_i = DLEQ((h, \tilde{\sigma}_i), (pk_i, \hat{\sigma}_i))$ (showing that the decrypted share $\tilde{\sigma}_i$ corresponds to $\hat{\sigma}_i$).
- **Amortized share decryption (for P_i):** If the PVSS has been used several times where P_i has received in each case a share $\hat{\sigma}_i^a$, P_i can decrypt shares as above but publish one single proof $PROOF_i = DLEQ((h, (\tilde{\sigma}_i^a)_a), (pk_i, (\hat{\sigma}_i^a)_a))$.
- **Share decryption verification:** Apply the verification algorithm of the DLEQ proof $PROOF_i$ and complain if this is not correct.
- **Secret reconstruction algorithm $Rec_{\mathcal{Q}}$:** On input $\{\tilde{\sigma}_i\}_{i\in\mathcal{Q}}$ for a set $\mathcal{Q}$ of exactly $n-t$ indices, for $j \in [\ell - 1]$:
  - Set $\lambda_i^{(j)} = \prod_{m:m\in\mathcal{Q},m\neq i} \frac{-j-m}{i-m}$ for all $i \in \mathcal{Q}$ and compute
$$S_j = \prod_{i\in\mathcal{Q}} (\tilde{\sigma}_i)^{\lambda_i^{(j)}} = \prod_{i\in\mathcal{Q}} h^{p(i)\lambda_i^{(j)}} = h^{p(-j)} = h^{s_j},$$
  - Publish the values S_j.
- **Reconstruction verification algorithm $RecVer_{\mathcal{Q}}$:** On input $(S_0, S_1, \ldots, S_{\ell-1}, \{\tilde{\sigma}_i\}_{i\in\mathcal{Q}})$, and calling $\mathcal{Q} = \{i_1, \ldots, i_{n-t}\}$ execute
$$Local_{LDEI}((\alpha_j)_{j\in[-(\ell-1),n-t]}, t+\ell-1, (\Sigma_j)_{j\in[-(\ell-1),n-t]}),$$
where $\alpha_j = j$ and $\Sigma_j = S_{-j}$ for $j \in [-(\ell-1), 0]$ and $\alpha_j = i_j$, $\Sigma_j = \tilde{\sigma}_{i_j}$ for $j \in [1, n-t]$.

Fig. 6. Reconstruction protocols and algorithms in π_{PPVSS}

These computations do not involve private information but especially in the PVSS they are expensive, requiring $O(n^2)$ exponentiations. Applying a resilient function via our $FFTE$ algorithm is considerably cheaper (it requires $O(n \log n)$ exponentiations), but depending on the application it still may make sense to apply the distributed computation techniques we are going to introduce.

On the other hand, given a purported output for such a computation, verifying their correctness can be done locally in a cheaper way ($O(n)$ exponentiations) using respectively the tests $Local_{LDEI}$ for verifying PVSS reconstruction and $Local_{LExp}$ (Appendix D) for verifying the correct application of $FFTE$.

In the worst case where $\Theta(n)$ parties abort after having correctly PVSSed their secrets, $\Theta(n)$ computations of each type need to be carried out. We balance the computational complexity of the parties as follows: for each of the tasks $\texttt{task}_i$ to be computed, a random set of computing parties A_i is chosen of cardinality around some fixed value $c(n)$, who independently compute the task and publish their claimed outputs; the remaining parties verify which one of the outputs is correct, and if none of them is, they compute the tasks themselves.

Remark 5. The choice of A_i has no consequences for the correctness and security of our protocols. The adversary may at most slow down the computation if it can arrange too many sets A_i to contain no honest parties, but this requires a considerable amount of biasing of the randomness source. We will derive this randomness using a random oracle applied to the transcript of the protocol up to that moment, and assume for simplicity that each party has probability roughly $c(n)/n$ to belong to each A_i.

Let $\mathcal{T} = \{\mathtt{task}_1, ..., \mathtt{task}_{f(n)}\}$ be a set of computation tasks, each of which consists of applying the same algorithm $AlgComp$ to an input $\mathtt{in}_i$. Likewise, let $AlgVer$ be a verifying algorithm that given an input $\mathtt{in}$ and a purported output $\mathtt{out}$ always accepts if the output is correct and rejects it with very large probability if it is incorrect. We apply the protocol in Figure 7.

Distributed computation protocol $DistComp(\mathcal{T}, \mathcal{P}, c(n))$

For each $i = 1, \ldots, f(n)$:
- A random subset $A_i \subseteq \mathcal{P}$ of $c(n)$ parties is selected.
- Each party $P_j \in A_i$ independently executes $AlgComp(\mathtt{in}_i)$ and publishes $\mathtt{out}_{P_j}$. Let $\mathcal{L}_i$ be the list of published claimed outputs for $\mathtt{task}_i$ ordered from most frequent (the one that is claimed to be the output by more parties in A_i) to least frequent.
- Each party $P_k \in \mathcal{P} \setminus A_i$ does the following
  - P_k applies $AlgVer(\mathtt{in}_i, \mathtt{out})$ for $\mathtt{out} \in \mathcal{L}_i$ in the order they appear in $\mathcal{L}_i$ until she finds a correct one, and accepts this as output of $\mathtt{task}_i$.
  - If none of the $\mathtt{out} \in \mathcal{L}_i$ passes the test, P_k computes $AlgComp(\mathtt{in}_i)$ and sets the result as output for $\mathtt{task}_i$.

Fig. 7. Distibuted computation protocol $DistComp(\mathcal{T}, \mathcal{P}, c(n))$

Computational complexity. We assume that $|\mathcal{P}| = \Theta(n)$, and that $AlgComp$ requires $\mathtt{ccost}(n)$ group exponentiations while $AlgVer$ needs $\mathtt{vcost}(n)$. On expectation, each party will participate as computing party for $O(f(n) \cdot c(n)/n)$ tasks and as verifier for the rest, in each case needing to verify at most $c(n)$ computations. Note that we schedule the verifications so that parties check first the most common claimed output, as this will likely be the correct one. For a given $\mathtt{task}_i$, if A_i contains at least one honest party, then one of the verifications will be correct. A_i contains only corrupt parties with probability $\tau^{c(n)}$ where $\tau = t/n$ and therefore we can assume that the number of i's for which this happens will be at most $O(\tau^{c(n)} f(n))$, so parties will need to additionally apply $AlgComp$ on this number of tasks. Therefore the number of exponentiations per party is

$$\mathtt{ccost}(n) \cdot O\left(\left(\frac{c(n)}{n} + \tau^{c(n)}\right) \cdot f(n)\right) + \mathtt{vcost}(n) \cdot O\left(c(n) \cdot \left(1 - \frac{c(n)}{n}\right) \cdot f(n)\right).$$

PVSS reconstruction. In the case of reconstruction of the PVSS'ed values, we have $AlgComp = Rec$ (Figure 6), which has complexity $\texttt{ccost}(n) = O(n^2)$ and $AlgVer$ is $RecVer$ where $\texttt{vcost}(n) = O(n)$. The number of computations $f(n)$ equals the number of corrupted parties that correctly share a secret but later decide not to reveal it. In the worst case $f(n) = \Theta(n)$. In that case, setting $c(n) = \log n$ gives a computational complexity of $O(n^2 \log n)$ exponentiations. In fact the selection $c(n) = \log n$ is preferable unless $f(n)$ is small ($f(n) = O(\log n)$) where $c(n) = n$ (everybody reconstructs the $f(n)$ computations independently) is a better choice. For the sake of simplicity we will use $c(n) = \log n$ in the description of the protocols.

Output reconstruction via $FFTE$. For this case we always have $f(n) = \ell = \Theta(n)$. We use $FFTE$ as $AlgComp$, so $\texttt{ccost}(n) = O(n \log n)$, while $AlgVer$ is $Local_{LExp}$ where $\texttt{vcost}(n) = O(n)$. Setting $c(n) = |\mathcal{P}|$, $c(n) = \log n$ or $c(n) = \Theta(1)$ all give $O(n^2 \log n)$ exponentiations in the worst case.

Setting $c(n) = \Theta(1)$ (a small constant number of parties computes each task, the rest verify) has a better best case asymptotic complexity: if every party acts honestly each party needs $O(n^2)$ exponentiations.

On the other hand, $c(n) = |\mathcal{P}|$ corresponds to every party carrying out the output computation by herself, so we do not really need $DistComp$. This needs less use of the ledger and a smaller round complexity, as the output of the majority is guaranteed to be correct. Moreover the practical complexity of $FFTE$ is very good, so in practice this option is computationally fast. We hence prefer this option, and state $c(n) = \Theta(1)$ as an alternative.

4.3 The ALBATROSS Multiparty Randomness Generation Algorithm

Next we present our randomness generation protocol ALBATROSS. We first introduce the following notation for having a matrix act on a matrix of group elements, by being applied to the matrix formed by their exponents.

Definition 5. *As above, let $\mathbb{G}_q$ be a group of order q, and h be a generator. Given a matrix $A = (A_{ij})$ in $\mathbb{Z}_q^{m_1 \times m_2}$ and a matrix $B = (B_{ij}) \in \mathbb{G}_q^{m_2 \times m_3}$, we define $C = A \diamond B \in \mathbb{G}_q^{m_1 \times m_3}$ with entries $C_{ij} = \prod_{k=1}^{m_2} B_{kj}^{A_{ik}}$.*

Remark 6. An alternative way to write this is $C = h^{A \cdot D}$, where D in $\mathbb{Z}_q^{m_2 \times m_3}$ is the matrix containing the discrete logs (in base h) of B, i.e. $D_{ij} = DLog_h(B_{ij})$. But we remark that we do not need to know D to compute C.

The protocol can be found in Figure 8 and Figure 9. In Figure 8 we detail the first two phases Commit and Reveal: in the Commit phase the parties share random tuples $(h^{s_0^a}, \ldots, h^{s_{\ell-1}^a})$ and prove correctness of the sharing. In the Reveal phase parties first verify correctness of other sharings. Once $n-t$ correct sharings have been posted,[4] the set $\mathcal{C}$ of parties that successfully posted correct sharings

[4]This is since $n - t$ is the maximum we can guarantee if t parties are corrupted. However we can also adapt our protocol to work with more than $n - t$ parties in $\mathcal{C}$ if these come before a given time limit.

now open the sharing polynomials. The remaining parties verify this is consistent with the encrypted shares. If all parties in $\mathcal{C}$ open secrets correctly, then all parties learn the exponents s_i^a and compute the final output by applying the resilient function in a very efficient manner, as explained in Figure 9, step 4'.

If some parties do not correctly open their secret tuples, the remaining parties will use the PVSS reconstruction routine to retrieve the values $h^{s_j^a}$, and then compute the final output from the reconstructed values, now computing the resilient functions in the exponent. This is explained in Figure 9.

Note that once a party gets into the set $\mathcal{C}$, her PVSS is correct (with overwhelming probability) and her tuple of secrets will be used in the final output, no matter the behaviour of that party from that point on. This is important: it prevents that the adversary biases the final randomness by initially playing honestly so that corrupted parties get into $\mathcal{C}$, and at that point deciding whether or not to open the secrets of each corrupted party conditioned on what other parties open. The fact that the honest parties can reconstruct the secrets from any party in $\mathcal{C}$ makes this behaviour useless to bias the output.

Theorem 2. *With overwhelming probability, the protocol Π_{ALB} has guaranteed output delivery and outputs a tuple of elements uniformly distributed in $\mathbb{G}_q^{\ell^2}$, as long as the active, static, computationally bounded adversary corrupts at most t parties (where $2t + \ell = n$).*

Proof. Guaranteed delivery is proved as mentioned above: once a party P_a gets into the set $\mathcal{C}$, we know that her sharing is correct w.o.p, because of the soundness of the LDEI proofs. No matter what the behaviour of P_a is from this point on, the secret tuple $(h^{s_0^a}, \ldots, h^{s_{\ell-1}^a})$ she shared in the PVSS will be a row of T. Since there are at most t corrupt parties, $\mathcal{C}$ is guaranteed to have $n - t$ parties, and every party in the protocol will learn the output $R = M \diamond T$.

Furthermore, to argue the output cannot be biased, note that parties in $\mathcal{C}$ can no longer change or erase their secrets after publishing the sharings, because even if parties refuse to open their secrets, these can be reconstructed in the Recovery phase. Therefore the matrix T is already determined when the set $\mathcal{C}$ is created. But at this point, the adversary has no information about other parties secrets because of the IND1-secrecy of the secret sharing scheme. The discrete logs of the secrets of each of the (at least) $n-2t$ honest parties in $\mathcal{C}$ are uniform vectors in $\mathbb{Z}_q^\ell$ in the view of the adversary at the point when the adversary publishes the sharings and LDEI-proofs. Therefore, the adversary can just influence on t rows of the matrix T without having any information about the other $n-2t$ rows. By the properties of the t-resilient function given by the matrix M (Definition 2) applied on every column of T, for every fixed selection of secrets the adversary makes the final output of the protocol is uniformly distributed.

Computational complexity: Group exponentiations. In Table 1 we collect the complexity of ALBATROSS in terms of number of group exponentiations per party, comparing it with the SCRAPE protocol, where for ALBATROSS we assume $\ell = \Theta(n)$. For the figures in the table, we consider both the worst case where $\Theta(n)$ parties in $\mathcal{C}$ do not open their secrets in the Reveal

Protocol Π_{ALB} (Commit and Reveal phases)

Protocol Π_{ALB} is run between a set $\mathcal{P}$ of n parties $P_1, \ldots, P_n$ who have access to a public ledger where they can post information for later verification. It is assumed that the Setup phase of π_{PPVSS} is already done and the public keys pk_i of each party P_i are already registered in the ledger. In addition, the parties have agreed on a Vandermonde $(n-2t) \times (n-t)$-matrix $M = M(\omega, n-2t, n-t)$ with $\omega \in \mathbb{Z}_q^*$ as specified in section 2.

1. **Commit:** For $1 \leq j \leq n$:
   - Party P_j executes the Distribution phase of the PVSS as Dealer for $\ell = n - 2t$ secrets, publishing the encrypted shares $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$ and sharing correctness verification information $LDEI^j$ on the public ledger, also learning the secrets $h^{s_0^j}, \ldots, h^{s_{\ell-1}^j}$ and the exponents $s_0^j, \ldots, s_{\ell-1}^j$.
2. **Reveal:**
   - For every set of encrypted shares $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$ and the verification information $LDEI^j$ published in the public ledger, all parties run the Verification phase of the PVSS sub protocol.
   - Once $n-t$ parties have posted a valid sharing on the ledger (we call $\mathcal{C}$ the set of these parties) each party $P_j \in \mathcal{C}$ reveals her sharing polynomial p_j.
   - Every party now verifies that indeed p_j is the sharing polynomial that P_j used in step 1 by reproducing the Distribution phase of P_j, i.e., computing the secrets s_i^j and shares σ_i^j of P_j, and verifying that $\hat{\sigma}_i^j$ is indeed equal to $pk_i^{\sigma_i^j}$. Note that at the same time they have computed the vector of secrets of P_j, i.e., $(s_0^j, \ldots, s_{\ell-1}^j)$.
   - At this point, if every party in $\mathcal{C}$ has opened their secrets correctly, go to step $4'$ in Figure 9. Otherwise proceed to step 3 in Figure 9

Fig. 8. Protocol Π_{ALB} (Commit and Reveal phases)

phase, and the best case where all the parties open their secrets. As we can see the amortized cost for generating a random group element goes down from $O(n^2)$ exponentiations to $O(\log n)$ in the first case and $O(1)$ in the second.

More in detail, in the Commit phase, both sharing a tuple of ℓ elements in the group costs $O(n)$ exponentiations and proving their correctness take $O(n)$ exponentiations. The Reveal phase takes $O(n^2)$ exponentiations since every party checks the $LDEI$ proofs of $O(n)$ parties, each costing $O(n)$ exponentiations, and similarly they later execute, for every party that reveals their sharing polynomial, $O(n)$ exponentiations to check that this is consistent with the encrypted shares.

In the worst case $O(n)$ parties from $\mathcal{C}$ do not open their secrets. The Recovery phase requires each then $O(n^2 \log n)$ exponentiations per party, as explained in Section 4.2. The Output phase also requires $O(n^2 \log n)$ exponentiations since $FFTE$ is used $O(n)$ times (or if the alternative distributed technique is used, the complexity is also $O(n^2 \log n)$ by the discussion in Section 4.2.

Protocol Π_{ALB} continued (Recovery and Output phase)

3 **Recovery:** Let $\mathcal{C}_A$ be the set of parties $P_a \in \mathcal{C}$ that do not publish the openings of their secrets in the Reveal phase, or that publish an erroneous opening.
- Every party $P_j \in \mathcal{P}$ executes the Amortized Share Decryption protocol for all PVSSs where a party $P_a \in \mathcal{C}_A$ was the dealer as described in Figure 6. That is, P_j posts all decrypted shares $\tilde{\sigma}_j^a$ and a unique $PROOF_j = DLEQ((h, (\tilde{\sigma}_j^a)_{P_a \in \mathcal{C}_A})(pk, (\hat{\sigma}_j^a)_{P_a \in \mathcal{C}_A}))$ to the public ledger.
- Each party $P_i \in \mathcal{P}$ verifies each proof $PROOF_j$ published by some P_j.
- Once a set $\mathcal{Q}$ of $n - t$ parties publish valid decrypted shares, the secrets are reconstructed as follows:
  For every $P_a \in \mathcal{C}_A$, we define $\texttt{task}_{Rec,a}$ to be the computation of $(h^{s_0^a}, \ldots, h^{s_{\ell-1}^a})$ from the decrypted shares with $AlgComp = Rec_{\mathcal{Q}}$ as described in PVSS reconstruction. Let $\mathcal{T}_{Rec} = \{\texttt{task}_{Rec,a}\}_{P_a \in \mathcal{C}_A}$.
  Parties call $DistComp(\mathcal{T}_{Rec}, \mathcal{P}, \log n)$, where $DistComp$ is as described in Figure 7 (where $AlgVer = RecVer_{\mathcal{Q}}$, as in Figure 6) using as randomness the output of a random oracle applied to the transcript so far.

4 **Output:** Let T be the $(n-t) \times \ell$ matrix with rows indexed by the parties in $\mathcal{C}$ and where the row corresponding to $P_a \in \mathcal{C}$ is $(h^{s_0^a}, \ldots, h^{s_{\ell-1}^a})$.
- Each computes the $\ell \times \ell$-matrix $R = M \diamond T$ by applying $FFTE$ to each column $T^{(j)}$ of T, resulting in column $R^{(j)}$ of R (since $R^{(j)} = M \diamond T^{(j)}$ and M is Vandermonde) for $j \in [0, \ell - 1]$. [a]
- Parties output the ℓ^2 elements of R as final randomness.

4' **Alternative output:** if every party in $\mathcal{C}$ has opened her secrets correctly in step **Reveal**, then:
- Parties compute $R = M \diamond T$ in the following way:
  Let S be the $(n-t) \times \ell$ matrix with rows indexed by the parties in $\mathcal{C}$ and where the row corresponding to $P_a \in \mathcal{C}$ is $(s_0^a, \ldots, s_{\ell-1}^a)$. Then each party computes $U = M \cdot S \in \mathbb{Z}_q^{\ell \times \ell}$ (using the standard FFT in $\mathbb{Z}_q$ to compute each column) and $R = h^U$. [b]
- Parties output the ℓ^2 elements of R as final randomness.

[a] Alternatively $DistComp$ can be used to distribute the computation, using committees of size $O(1)$ to compute each column and $Local_{LExp}$ (Appendix D) to verify these computations, see discussion in Section 4.2

[b] Meaning the (i, j)-th element in R is h^y where y is the (i, j)-th element in U

Fig. 9. Protocol Π_{ALB} continued

In the best case, all parties from $\mathcal{C}$ reveal their sharing polynomials correctly, the Recovery phase is not necessary and the Output phase requires $O(n^2)$ exponentiations per party as parties can compute the result directly by reconstructing the exponents first (where in addition one can use the standard FFT in $\mathbb{Z}_q$).

Computational complexity: Other operations. The total number of additional computation of group operations (aside from the ones involved in computing group exponentiations) is $O(n^2 \log n)$. With regard to operations in the field $\mathbb{Z}_q$, parties need to carry out a total of $O(n)$ computations of poly-

Scheme	Output size	Complexity(# group exponentiations)					Amortized complexity
		Commit	Reveal	Recovery	Output	Total	
SCRAPE	1	$O(n)$	$O(n^2)$	$O(n)$	$O(1)$	$O(n^2)$	$O(n^2)$
ALBATROSS, worst case	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n^2 \log n)$	$O(n^2 \log n)$	$O(n^2 \log n)$	$O(\log n)$
ALBATROSS, best case	$O(n^2)$	$O(n)$	$O(n^2)$	-	$O(n^2)$	$O(n^2)$	$O(1)$

Table 1. Computational complexity in terms of numbers of exponentiations for each phase of the protocols, and exponentiations per created element (per party).

nomials of degree $O(n)$ in sets of $O(n)$ points, which are always subsets of the evaluation points for the secrets and share. In order to speed this computation up we can use $2n - th$ roots of unity as evaluation points (instead of $[-\ell - 1, n]$) and make use of the FFT yielding a total of $O(n^2 \log n)$ basic operations in $\mathbb{Z}_q$. We also need to compute Lagrange coefficients and the values u_i in $Local_{LDEI}$ but this is done only once per party. In addition, the recent article [42] has presented efficient algorithms for all these computations.

5 Making ALBATROSS Universally Composable

In the previous sections, we constructed a packed PVSS scheme π_{PPVSS} and used it to construct a guaranteed output delivery (G.O.D.) randomness beacon Π_{ALB}. However, as in previous G.O.D. unbiasable randomness beacons [31,17], we only argue stand alone security for this protocol. In the remainder of this work, we show that Π_{ALB} can be lifted to achieve Universally Composability by two different approaches: 1. using UC-secure zero knowledge proofs of knowledge for the LDEI and DLEQ relations defined above, and 2. using UC-secure additively homomorphic commitments. We describe the UC framework, ideal functionalities and additional modelling details in Appendix E.

Modeling Randomness Beacons in UC We are interested in realizing a publicly verifiable G.O.D. coin tossing ideal functionality that functions as a randomness beacon (*i.e.* it allows any third party verifier to check whether a given output was previously generated by the functionality). We define such a functionality $\mathcal{F}_{\mathsf{CT}}^{m,\mathcal{D}}$ in Figure 10. Notice that it provides random outputs once all honest parties activate it with (Toss, *sid*) independently from dishonest parties' behavior. We realize this simple functionality for single shot coin tossing because it allows us to focus on the main aspects of our techniques. In order to obtain a stream of random values as in a traditional beacon, all parties can periodically call this functionality with a fresh *sid*.

5.1 Using UC-secure Zero Knowledge Proofs

Our first approach is to modify the commit and reveal phases of Protocol Π_{ALB} and use NIZK ideal functionalities as setup (apart from $\mathcal{F}_{APBB}$ as defined in

Functionality $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$

$\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ is parameterized by $k \in \mathbb{N}$ and a distribution $\mathcal{D}$, interacting with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ through the following interfaces:

Toss: Upon receiving (TOSS, *sid*) from all honest parties in $\mathcal{P}$, uniformly sample k random elements $x_1, \ldots, x_k \xleftarrow{\$} \mathcal{D}$ and send (TOSSED, $sid, x_1, \ldots, x_k$) to all parties in $\mathcal{P}$.
Verify: Upon receiving (VERIFY, $sid, x_1, \ldots, x_k$) from $\mathcal{V}_j \in \mathcal{V}$, if (TOSSED, $sid, x_1, \ldots, x_k$) has been sent to all parties in $\mathcal{P}$ set $f = 1$, otherwise, set $f = 0$. Send (VERIFIED, $sid, x_1, \ldots, x_k, f$) to $\mathcal{V}_j$.

Fig. 10. Functionality $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ for G.O.D. Publicly Verifiable Coin Tossing.

Appendix E) in order to obtain an UC-secure version of protocol. The crucial difference is that instead of having all parties reveal the randomness of the PVSS sharing algorithm (*i.e.* the polynomial $p(X)$) in the reveal phase in order to verify that certain random inputs were previously shared in the commit phase, we have the parties commit to their random inputs using an equivocal commitment and then generate a NIZK proof that the random inputs in the commitments correspond to the ones shared by the PVSS scheme in the commit phase. In the reveal phase, the parties simply open their commitments. In case a commitment is not opened, the honest parties use the PVSS reconstruction to recover the random input. Intuitively, using an equivocal commitment scheme and ideal NIZKs allows the simulator to first extract all the random inputs shared by the adversary and later equivocate the simulated parties' commitment openings in order to trick the adversary into accepting arbitrary random inputs from simulated honest parties that result in the same randomness as obtained from $\mathcal{F}_{\mathsf{CT}}$. Protocol Π_{CT-ZK} is presented in Figures 11 and 12.

Pedersen Commitments Another component we will use is a Pedersen commitment [36], which is an *equivocal commitment*, *i.e.* it allows a simulator who knows a trapdoor to open a commitment to any arbitrary message. In this scheme, all parties are assumed to know generators g, h of a group $\mathbb{G}_q$ of prime order q chosen uniformly at random such that the discrete logarithm of h on base g is unknown. In order to commit to a message $m \in \mathbb{Z}_q$, a sender samples a randomness $r \xleftarrow{\$} \mathbb{Z}_q$ and computes a commitment $c = g^m h^r$, which can be later opened by revealing (m, r). In order to verify that an opening (m', r') for a commitment c is valid, a receiver simply checks that $c = g^{m'} h^{r'}$. However, a simulator who knows a trapdoor td such that $h = g^{\mathsf{td}}$ can open $c = g^m h^r$ to any arbitrary message m' by computing $r' = \frac{m + \mathsf{td} \cdot r - m'}{\mathsf{td}}$ and revealing (m', r'). For a message $m \in \mathbb{Z}_q$ and randomness $r \in \mathbb{Z}_q$, we denote a commitment c as $\mathsf{Com}(m, r)$, the opening of c as $\mathsf{Open}(m, r)$ and the opening of c to an arbitrary message $m' \in \mathbb{Z}_q$ given trapdoor td as $\mathsf{TDOpen}(m, r, m', \mathsf{td})$.

Protocol Π_{CT-ZK} (Initialization, Commit and Reveal)

It is assumed that $\mathcal{F}_{\mathsf{CRS}}$ provides Pedersen commitment parameters $g_p, h_p \in \mathbb{G}_q$ and a Vandermonde $(n-2t) \times (n-t)$-matrix $M = M(\omega, n-2t, n-t)$ with $\omega \in \mathbb{Z}_q^*$ as specified in section 2. We denote the commitment and open procedures of a Pedersen commitment as $\mathsf{Com}(m, r)$ and $\mathsf{Open}(m, r)$, respectively. Protocol Π_{CT-ZK} is run between a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ (out of which at most t are corrupted) and a set of verifiers $\mathcal{V}$ interacting with each other and with functionalities $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{NIZK}}^{LDEI}, \mathcal{F}_{\mathsf{NIZK}}^{DLEQ}, \mathcal{F}_{\mathsf{NIZK}}^{COMC}$ as follows:

1. **Initialization:** Upon being activated for the first time, all parties in $\mathcal{P}$ and $\mathcal{V}$ send (CRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$, obtaining $(\text{CRS}, sid, g_p, h_p, M)$. Each party $\mathcal{P}_i \in \mathcal{P}$ samples $sk_i \leftarrow \mathbb{Z}_q$, computes $pk_i = h^{sk_i}$ and sends $(\text{POST}, sid, \text{MID}, pk_i)$ to $\mathcal{F}_{APBB}$ using a fresh MID. Finally, all parties obtain all pk_i from $\mathcal{F}_{APBB}$.
2. **Commit:** For $1 \leq j \leq n$:
   (a) Party P_j executes the Distribution phase of of π_{PPVSS} (Figure 5) as Dealer for $\ell = n-2t$ random inputs using $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ to compute the NIZKs, obtaining encrypted shares $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$, a NIZK proof π_{LDEI}^j, secrets $h^{s_0^j}, \ldots, h^{s_{\ell-1}^j}$ and exponents $s_0^j, \ldots, s_{\ell-1}^j$.
   (b) P_j computes $\mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j)$ (with fresh randomness $r_0^j, \ldots, r_{\ell-1}^j \leftarrow \mathbb{Z}_q$) and obtains from $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$ a NIZK proof π_{COMC}^j that these commitments contain the same secrets $s_0^j, \ldots, s_{\ell-1}^j$ as $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$.
   (c) P_j sends $(\text{POST}, sid, \text{MID}, (\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j, \pi_{LDEI}^j, \mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j), \pi_{COMC}^j))$ to $\mathcal{F}_{APBB}$ using a fresh MID.
3. **Reveal:**
   (a) All parties in $\mathcal{P}$ send (READ, sid) to $\mathcal{F}_{APBB}$, receive $(\text{READ}, sid, \mathcal{M})$ and, for every new $(\mathcal{P}_i, sid, \text{MID}, (\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j, \pi_{LDEI}^j, \mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j), \pi_{COMC}^j))$ in $\mathcal{M}$, verify proof π_{COMC}^j using $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$ and run the Verification phase of π_{PPVSS} (Figure 5) using $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$.
   (b) Once $n-t$ parties have posted valid $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$, π_{LDEI}^j and $\mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j), \pi_{COMC}^j$ on $\mathcal{F}_{APBB}$ (we call $\mathcal{C}$ the set of these parties) each party $P_j \in \mathcal{C}$ sends $(\text{POST}, sid, \text{MID}, \big(\mathsf{Open}(s_0^j, r_{0,j}), \ldots, \mathsf{Open}(s_{\ell-1}^j, r_{\ell-1,j})\big))$ to $\mathcal{F}_{APBB}$ using a fresh MID, for $j \in \mathcal{C}$.
   (c) All parties in $\mathcal{P}_i$ send (READ, sid) to $\mathcal{F}_{APBB}$, receive $(\text{READ}, sid, \mathcal{M})$ and check that $(\mathcal{P}_i, sid, \text{MID}, \big(\mathsf{Open}(s_0^j, r_{0,j}), \ldots, \mathsf{Open}(s_{\ell-1}^j, r_{\ell-1,j})\big))$ is in $\mathcal{M}$ for all $j \in \mathcal{C}$. Once this check succeeds, all parties in $\mathcal{P}$ verify that these correspond to the secrets that were shared, by computing all $h^{s_i^j}$ and checking the consistency of these values with the published shares with the check $Local_{LDEI}$, in the same way that they would do in Figure 6.
   (d) If any of the checks in the previous step fails, proceed to the recovery phase of Figure 12. Otherwise, if every party in $\mathcal{C}$ has opened their secrets correctly, parties compute $R = M \diamond T$ as follows. Let S be the $(n-t) \times \ell$ matrix with rows indexed by the parties in $\mathcal{C}$ and where the row corresponding to $P_a \in \mathcal{C}$ is $(s_0^a, \ldots, s_{\ell-1}^a)$. All parties in $\mathcal{P}$ compute $U = M \cdot S \in \mathbb{Z}_q^{\ell \times \ell}$ and $R = h^U$, outputting the ℓ^2 elements of R as final randomness.

Fig. 11. Protocol Π_{CT-ZK}, optimistic case (Initialization, Commit and Reveal).

Protocol Π_{CT-ZK} continued, pessimistic case (Recovery phase)

4 **Recovery:** Let $\mathcal{C}_A$ be the set of parties $P_a \in \mathcal{C}$ that do not publish a valid opening of their commitments in the reveal phase. Every party $\mathcal{P}_j \in \mathcal{P}$ proceed as follows:

(a) Execute the Share Decryption protocol for each PVSS where a party $P_a \in \mathcal{C}_A$ was the dealer as described in Figure 6 using $\mathcal{F}_{\mathsf{NIZK}}^{DLEQ}$ to compute π_{DLEQ}^j. $\mathcal{P}_j$ sends $(\text{POST}, sid, \text{MID}, (\{\tilde{\sigma}_j^a\}_{P_a \in \mathcal{C}_A}, \pi_{DLEQ}^j))$ to $\mathcal{F}_{APBB}$ using a fresh MID.

(b) Send (READ, sid) to $\mathcal{F}_{APBB}$, receive $(\text{READ}, sid, \mathcal{M})$ and, for every new $(\mathcal{P}_i, sid, \text{MID}, (\{\tilde{\sigma}_j^a\}_{P_a \in \mathcal{C}_A}, \pi_{DLEQ}^j))$ in $\mathcal{M}$, verify proof π_{DLEQ}^j using $\mathcal{F}_{\mathsf{NIZK}}^{DLEQ}$.

(c) Once a set $\mathcal{Q}$ of $n - t$ parties have posted valid decrypted shares on $\mathcal{F}_{APBB}$, the secrets are reconstructed as follows. For every $P_a \in \mathcal{C}_A$, we define $\mathtt{task}_{Rec,a}$ to be the computation of $(h^{s_0^a}, \ldots, h^{s_{\ell-1}^a})$ from the decrypted shares with $Rec_{\mathcal{Q}}$ as described in PVSS reconstruction. Let $\mathcal{T}_{Rec} = \{\mathtt{task}_{Rec,a}\}_{P_a \in \mathcal{C}_A}$. Then call $DistComp(\mathcal{T}_{Rec}, \mathcal{P}, \log n)$, where $DistComp$ is as described in Figure 7 with $AlgComp = Rec_{\mathcal{Q}}$ and $AlgVer = RecVer_{\mathcal{Q}}$ (Figure 6), taking all inputs from $\mathcal{F}_{APBB}$ and posting all outputs to $\mathcal{F}_{APBB}$.

(d) Send (READ, sid) to $\mathcal{F}_{APBB}$, obtaining $\mathcal{M}$. Let T be the $(n-t) \times \ell$ matrix with rows indexed by the parties in $\mathcal{C}$ and where the row corresponding to $P_a \in \mathcal{C}$ is $(h^{s_0^a}, \ldots, h^{s_{\ell-1}^a})$, which are obtained from $\mathcal{M}$.

(e) Each computes the $\ell \times \ell$-matrix $R = M \diamond T$ by applying $FFTE$ to each column $T^{(j)}$ of T, resulting in column $R^{(j)}$ of R (since $R^{(j)} = M \diamond T^{(j)}$ and M is Vandermonde) for $j \in [0, \ell - 1]$.

(f) Output the ℓ^2 elements of R as final randomness.

5 **Verify:** On input $(\text{VERIFY}, sid, x_1, \ldots, x_k)$, a verifier $\mathcal{V}_i \in \mathcal{V}$ checks that the protocol transcript registered in $\mathcal{F}_{APBB}$ is valid using the verification interfaces of $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}, \mathcal{F}_{\mathsf{NIZK}}^{DLEQ}, \mathcal{F}_{\mathsf{NIZK}}^{COMC}$. If the transcript is valid and results in output $x_1, \ldots, x_k$, $AlgVer_i$ sets $b = 1$, else, it sets $b = 0$. $\mathcal{V}_i$ outputs $(\text{VERIFIED}, sid, x_1, \ldots, x_k, b)$.

Fig. 12. Protocol Π_{CT-ZK} continued, pessimistic case (Recovery phase)

NIZKs We use three instances of functionality $\mathcal{F}_{\mathsf{NIZK}}^{R}$. The first one is $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$, which is parameterized with relation $LDEI$ (Section 3). The second one is $\mathcal{F}_{\mathsf{NIZK}}^{DLEQ}$, which is parameterized with relation $DLEQ$ for multiple statements $DLEQ((h, (\tilde{\sigma}_j^i)_{i \in I})(pk, (\hat{\sigma}_j^i)_{i \in I}))$ (Section 3). The third and final one is $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$, which is parameterized with a relation $COMC$ showing that commitments $\mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j)$ contain the same secrets $s_0^j, \ldots, s_{\ell-1}^j$ as in the encrypted shares $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$ generated by π_{PPVSS} (Figure 5).

CRS and Bulletin Board In order to simplify our protocol description and security analysis, we assume that parties have access to a CRS containig the public parameters for the Pedersen equivocal commitment scheme and Vandermonde

matrix for the PVSS scheme π_{PPVSS}. Moreover, a CRS would be necessary to realize the instances of $\mathcal{F}_{\mathsf{NIZK}}^{R}$ we use. Nevertheless, we remark that the parties could generate all of these values in a publicly verifiable way through a multiparty computation protocol [9] and register them in the authenticated public bulletin board functionality in the beginning of the protocol.

Communication Model Formally, for the sake of simplicity, we describe our protocol using an ideal authenticated public bulletin board $\mathcal{F}_{APBB}$ that guarantees all messages appear immediately in the order they are received and become immutable. However, we remark that our protocols can be proven secure in a semi-synchronous communication model with a public ledger where messages are arbitrarily delayed and re-ordered by the adversary but eventually registered (*i.e.* the adversary cannot drop messages or induce an infinite delay). Notice that the protocol proceeds to each of its steps once $n - t$ parties (*i.e.* at least all honest parties) post their messages to $\mathcal{F}_{APBB}$, so it is guaranteed to terminate if honest party messages are delivered eventually regardless of the order in which these messages appear or of the delay for such messages to become immutable. Using the terminology of [26,3], if we were to use a blockchain based public ledger instead of $\mathcal{F}_{APBB}$, each point we state that the parties wait for $n-t$ valid messages to be posted to $\mathcal{F}_{APBB}$ could be adapted to having the parties wait for enough rounds such that it is guaranteed by the chain growth property that a large number enough blocks are added to the ledger in such a way that the chain quality property guarantees that at least one of these blocks is honest (*i.e.* containing honest party messages) and that enough blocks are guaranteed to be added after this honest block so that the common prefix property guarantees that all honest parties have this block in their local view of the ledger. A similar analysis has been done in [31,23] in their constructions of randomness beacons.

Complexity We execute essentially the same steps of Protocol Π_{ALB} with the added overhead of having each party compute Pedersen Commitments to their secrets and generate a NIZK showing these secrets are the same as the ones shared through the PVSS scheme. Using the combined approaches of [12,33] to obtain these NIZKs, the approximate extra overhead of using UC NIZKs in relation to the stand alone NIZKs of Π_{ALB} will be that of computing 2 evaluations of the Paillier cryptosystem's homomorphism and 4 modular exponentiations over $\mathbb{G}_q$ per each secret value in the witness for each NIZK. In the Commit and Reveal phases, this yields an approximate fixed extra cost of $4n^2$ evaluations of the Paillier cryptosystem's homomorphism and $8n^2$ modular exponentiations over $\mathbb{G}_q$ for generating and verifying NIZKs with $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ and $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$. In the recovery phase, if a parties fail to open their commitments, there is an extra costs of $2a(n-t)$ evaluations of the Paillier cryptosystem's homomorphism and $4a(n-t)$ modular exponentiations over $\mathbb{G}_q$ for generating and verifying NIZKs with $\mathcal{F}_{\mathsf{NIZK}}^{DLEQ}$. In terms of communication, the approximate extra overhead is of one Paillier ciphertext and two integer commitments per each secret value in the witness for each NIZK, yielding an approximate total overhead of

$(n^2 + a(n-t)) \cdot |\mathsf{Paillier}| + (2n^2 + a(n-t)) \cdot |\mathbb{G}_q|$ bits where $|\mathsf{Paillier}|$ is the length of a Paillier ciphertext and $|\mathbb{G}_q|$ is the length of a $\mathbb{G}_q$ element.

Theorem 3. *Protocol Π_{CT-ZK} UC-realizes $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ for $k = \ell^2 = (n-2t)^2$ and $\mathcal{D} = \{h^s | h \in \mathbb{G}_q, s \xleftarrow{\$} \mathbb{Z}_q\}$ in the $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{NIZK}}^{LDEI}, \mathcal{F}_{\mathsf{NIZK}}^{DLEQ}, \mathcal{F}_{\mathsf{NIZK}}^{COMC}$-hybrid model with static security against an active adversary $\mathcal{A}$ corrupting corrupts at most t parties (where $2t + \ell = n$) parties under the DDH assumption.*

Proof. We prove this theorem in Appendix G.

5.2 Using Designated Verifier Homomorphic Commitments

In the stand alone version of ALBATROSS and the first UC-secure version we construct, the main idea is to encrypt shares of random secrets obtained from packed Shamir secret sharing and prove in zero knowledge that those shares were consistently generated. Later on, zero knowledge proofs are used again to prove that decrypted were correctly obtained from the ciphertexts that have already been verified for consistency, ensuring secrets can be properly reconstructed. We now explore an alternative where we instead commit to their shares using a UC additively homomorphic commitment scheme and perform a version the $Local_{LDEI}$ check on the committed shares and open the resulting commitment in order to prove that their shares were correctly generated. In order to do that, we need a new notion of a UC additively homomorphic commitment that allows for the sender to open a commitments to an specific share towards a specific party (so that only that party learns its share) but allows for those parties to later prove that they have received a valid opening or not, allowing the other parties to reconstruct the secrets from the opened shares. In the remainder of this section, we introduce our new definition of such a commitment scheme and show how it can be used along with $\mathcal{F}_{APBB}$ to realize $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$.

Designated Verifier Commitments We define a new flavor of multi-receiver commitments that we call Designated Verifier Commitments, meaning that they allow a sender to open a certain commitment only towards a certain receiver in such a way that this receiver can later prove that the commitment was correctly opened (also revealing its message) or that the opening was not valid. Moreover, we give this commitments the ability to evaluate linear functions on committed values and reveal only the result of these evaluations but not the individual values used as input, a property that is called additive homomorphism. We depart from the multi-receiver additively homomorphic commitment functionality from [15] and augment it with designated verifier opening and verification interfaces. Functionality $\mathcal{F}_{\text{DVHCOM}}$ is presented in Figure 13. The basic idea to realize this functionality is that we make two important changes to the protocol of [15]: 1. all protocol messages are posted to the authenticated bulletin board $\mathcal{F}_{APBB}$; 2. designated openings are done by encrypting the opening information from the protocol of [15] with the designated verifier's public key for a cryptosystem with plaintext verification [5], which allows the designated verifier to later publicly

Functionality $\mathcal{F}_{\text{DVHCOM}}$

$\mathcal{F}_{\text{DVHCOM}}$ keeps two initially empty lists $\mathsf{open}_{\mathsf{des}}$ and $\mathsf{open}_{\mathsf{pub}}$. $\mathcal{F}_{\text{DVHCOM}}$ interacts with a sender P_S, a set of receivers $P = \{P_1, \ldots, P_t\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ and proceeds as follows:

- **Commit Phase**: The length of the committed messages λ is fixed and known to all parties.
  - Upon receiving a message $(\text{COMMIT}, sid, ssid, P_S, P, \boldsymbol{m})$ from P_S, where $\boldsymbol{m} \in \{0,1\}^\lambda$, record the tuple $(ssid, P_S, P, \boldsymbol{m})$ and send the message $(\text{RECEIPT}, sid, ssid, P_S, P)$ to every receiver $P_i \in P$ and $\mathcal{S}$. Ignore any future commit messages with the same $ssid$ from P_S to P.
  - If a message (ABORT, sid) is received from $\mathcal{S}$, the functionality halts.
- **Addition**: Upon receiving a message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_S, P)$ from P_S: If tuples $(ssid_1, P_S, P, \boldsymbol{m}_1)$, $(ssid_2, P_S, P, \boldsymbol{m}_2)$ were previously recorded and $ssid_3$ is unused, record $(ssid_3, P_S, P, \boldsymbol{m}_1 + \boldsymbol{m}_2)$ and send the message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_S, P, \mathsf{success})$ to P_S, every $P_i \in P$ and $\mathcal{S}$.
- **Schedule Public Open**: Upon receiving a message $(\mathsf{P-Open}, sid, ssid)$ from P_S, if a tuple $(ssid, P_S, P, \boldsymbol{m})$ was previously recorded, append $ssid$ to $\mathsf{open}_{\mathsf{pub}}$.
- **Schedule Designated Open**: Upon receiving a message $(\mathsf{D-Open}, sid, P_d, ssid)$ from P_S for $P_d \in P$, if a tuple $(ssid, P_S, P, \boldsymbol{m})$ was previously recorded, append $(P_d, ssid)$ to $\mathsf{open}_{\mathsf{des}}$.
- **Execute Open**: Upon receiving a message $(\mathsf{Do-Open}, sid)$ from P_S:
  - For every $ssid \in \mathsf{open}_{\mathsf{pub}}$, send $(\text{P-REVEAL}, sid, P_S, P, ssid, \boldsymbol{m})$ to every receiver $P_i \in P$ where $\boldsymbol{m}$ is in the recorded tuple $(ssid, P_S, P, \boldsymbol{m})$.
  - For every pair $(P_d, ssid) \in \mathsf{open}_{\mathsf{des}}$ send $(\text{D-REVEAL}, sid, P_S, P_d, ssid)$ to every receiver in P and send $(\text{D-REVEAL}, sid, P_S, P_d, ssid, \boldsymbol{m})$ to P_d where $\boldsymbol{m}$ is in the recorded tuple $(ssid, P_S, P, \boldsymbol{m})$.

  Stop responding to $\mathsf{P-Open}$, $\mathsf{D-Open}$ and $\mathsf{Do-Open}$ queries.
- **Reveal Designated Open** Upon receiving message $(\text{REVEAL-D-OPEN}, sid, P_d, ssid)$ from P_d, if $(P_d, ssid) \in \mathsf{open}_{\mathsf{des}}$ and Execute Open has happened, send $(\text{P-REVEAL}, sid, P_S, P, ssid, \boldsymbol{m})$ to every receiver $P_i \in P$ where $\boldsymbol{m}$ is in the recorded tuple $(ssid, P_S, P, \boldsymbol{m})$.
- **Verify** Upon receiving $(\text{VERIFY}, sid, ssid, P_S, \boldsymbol{m})$ from $\mathcal{V}_j \in \mathcal{V}$, if $(\text{P-REVEAL}, sid, P_S, P, ssid, \boldsymbol{m})$ was sent to every receiver $P_i \in P$, set $f = 1$, else, set $f = 0$. Send $(\text{VERIFIED}, sid, ssid, P_S, \boldsymbol{m}, f)$ to $\mathcal{V}_j$.

Fig. 13. Functionality $\mathcal{F}_{\text{DVHCOM}}$

prove that a certain (in)valid commitment opening was in the ciphertext. Interestingly, $\mathcal{F}_{\text{DVHCOM}}$ can be realized in the global random oracle model under the Computational Diffie Hellman (CDH) assumption. We show how to realize $\mathcal{F}_{\text{DVHCOM}}$ in Appendix F.

Realizing $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{P}}$ with Π_{CT-COM} The main idea in constructing Protocol Π_{CT-COM} is to have each party compute shares of their random secrets using packed Shamir secret sharing and then generate designated verifier commitments $\mathcal{F}_{\text{DVHCOM}}$ to each share. Next, each party proves that their committed shares a valid by executing the $Local_{LDEI}$ test on the committed shares (instead

Protocol Π_{CT-COM}

Let $\ell = n - 2t$. We assume the parties have a Vandermonde $(\ell) \times (n-t)$-matrix $M = M(\omega, \ell, n-t)$ with $\omega \in \mathbb{Z}_q^*$ as specified in section 2. Protocol Π_{CT-COM} is run between a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ (out of which at most t are corrupted) and a set of verifiers $\mathcal{V}$ interacting with each other and with functionalities $\mathcal{F}_{APBB}, \mathcal{F}_{\text{DVHCOM}}$ as follows:

1. **Commit:** On input (TOSS, sid), every party $\mathcal{P}_i \in \mathcal{P}$ proceeds as follows:
   (a) $\mathcal{P}_i$ acts as dealer in Shamir packed secret sharing, sampling a polynomial $p(X) \leftarrow \mathbb{Z}_q[X]_{\leq t+\ell-1}$ such that $s_0 = p(0), s_1 = p(-1), \ldots, s_{\ell-1} = p(-(\ell-1))$ and computing shares $\sigma_i = p(i)$ for $1 \leq i \leq n$.
   (b) For $1 \leq j \leq n$, $\mathcal{P}_i$ picks an unused $ssid_j^i$ and sends (COMMIT, $sid, ssid_j, \mathcal{P}_i, \mathcal{P}, \sigma_j$) to $\mathcal{F}_{\text{DVHCOM}}$.
   (c) $\mathcal{P}_i$ uses the Addition interface of $\mathcal{F}_{\text{DVHCOM}}$ to evaluate the $Local_{LDEI}$ test on the committed shares identified by $ssid_1^i, \ldots, ssid_n^i$ obtaining a new commitment identified by $ssid_{LDEI}^i$. The random polynomial used by $Local_{LDEI}$ is sampled via de Fiat-Shamir heuristic using the output of a global random oracle queried on the protocol transcript so far.
   (d) $\mathcal{P}_i$ sends $(\mathsf{P-Open}, sid, ssid_{LDEI}^i)$ to $\mathcal{F}_{\text{DVHCOM}}$ (scheduling a public opening the commitment with the $Local_{LDEI}$ result) and, for $1 \leq j \leq n$, sends $(\mathsf{D-Open}, sid, \mathcal{P}_j, ssid_j^i)$ to $\mathcal{F}_{\text{DVHCOM}}$ (scheduling the delegated opening of share σ_j towards $\mathcal{P}_j$). Finally, $\mathcal{P}_i$ sends $(\mathsf{Do-Open}, sid)$ to $\mathcal{F}_{\text{DVHCOM}}$ execute all openings and sends (POST, sid, MID, $\boldsymbol{m}_{LDEI}^i$) to $\mathcal{F}_{APBB}$ using a fresh MID (registering the result of the LDEI test on the bulletin board).
   (e) For $1 \leq j \leq n$, $\mathcal{P}_i$ checks that it has received (P-REVEAL, $sid, \mathcal{P}_j, \mathcal{P}, ssid_{LDEI}^j, 0$) (meaning that the shares from $\mathcal{P}_j$ passed the $Local_{LDEI}$ test), (D-REVEAL, $sid, \mathcal{P}_j, \mathcal{P}_i, ssid_i^j, \sigma_i^j$) and (D-REVEAL, $sid, \mathcal{P}_j, \mathcal{P}_i, ssid_{j'}^j$) for every $j' = 1, \ldots, n, j' = j$ (meaning that $\mathcal{P}_j$ opened each committed share towards the right designated verifier) from $\mathcal{F}_{\text{DVHCOM}}$. We call the set of parties for which this check succeeds $\mathcal{C}$, which is guaranteed to contain at least $n - t$ parties (all honest parties).
2. **Reveal and Output:** Every party $\mathcal{P}_i \in \mathcal{P}$ proceeds as follows:
   (a) For every party $\mathcal{P}_j \in \mathcal{C}$, $\mathcal{P}_i$ sends (REVEAL-D-OPEN, $sid, \mathcal{P}_i, ssid_i^j$) to $\mathcal{F}_{\text{DVHCOM}}$ and (POST, sid, MID, σ_i^j) to $\mathcal{F}_{APBB}$ using a fresh MID.
   (b) After the $n - t$ honest parties open their committed shares, perform the recovery procedure of Π_{ALB} directly on the set of shares σ_o^j such that $\mathcal{P}_j \in \mathcal{C}$ and $\mathcal{P}_o$ revealed its shares in the previous step (which is guaranteed to contain at least $n-t$ shares revealed by the honest parties). Output the ℓ^2 elements of R as final randomness.
3. **Verify:** On input (VERIFY, $sid, x_1, \ldots, x_k$), a verifier $\mathcal{V}_i \in \mathcal{V}$ checks that the protocol transcript registered in $\mathcal{F}_{APBB}$ is valid using the verification interface of $\mathcal{F}_{\text{DVHCOM}}$. If the transcript is valid and results in output $x_1, \ldots, x_k$, $AlgVer_i$ sets $b = 1$, else, it sets $b = 0$. $\mathcal{V}_i$ outputs (VERIFIED, $sid, x_1, \ldots, x_k, b$).

Fig. 14. Protocol Π_{CT-COM}.

of group exponents), which involves evaluating a linear function on the committed shares and publicly opening the commitment containing the result of this evaluation. At the same time, each party performs designated openings of each committeed share towards one of the other parties, who verify that they have obtained a valid designated opening and post a message to $\mathcal{F}_{APBB}$ confirming that this check succeeded. After a high enough number of parties successfully confirms this check for each of the sets of committed shares, each party publicly opens all of their committed shares, allowing the other parties to reconstruct the secrets. If one of the parties does not open all of their shares, the honest parties can still reconstruct the secrets by revealing the designated openings they received for their shares. We present Protocol Π_{CT-COM} in Figure 14 and state its security in Theorem 4. Since $\mathcal{F}_{\text{DVHCOM}}$ can be realized in the global random oracle model under the Computational Diffie Hellman (CDH) assumption as shown in Appendix F, we obtain an instantiation of $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ with security based on CDH.

Theorem 4. *Protocol Π_{CT-COM} UC-realizes $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ for $k = \ell^2 = (n-2t)^2$ and $\mathcal{D} = \{h^s | h \in \mathbb{G}_q, s \xleftarrow{\$} \mathbb{Z}_q\}$ in the $\mathcal{F}_{\text{DVHCOM}}, \mathcal{F}_{APBB}$-hybrid model with static security against an active adversary $\mathcal{A}$ corrupting at most t parties (where $2t + \ell = n$).*

Proof. Constructing a simulator $\mathcal{S}$ to prove this theorem is simple due to the use of $\mathcal{F}_{\text{DVHCOM}}$. $\mathcal{S}$ follows the instructions of an honest party in executing Π_{CT-COM} with an internal copy $\mathcal{A}$ of the adversary simulating $\mathcal{F}_{\text{DVHCOM}}, \mathcal{F}_{APBB}$ towards $\mathcal{A}$. After the Commitment Phase is completed, the set $\mathcal{C}$ contains all the parties who correctly committed to valid shares (as checked by the $Local_{LDEI}$ procedure) and did designated openings of each share towards the right party. This guarantees that the honest parties can always reveal their designated openings publicly, revealing their shares so that the secrets of all parties in $\mathcal{C}$ can always be reconstructed. $\mathcal{S}$ extracts the shares in commitments by parties in $\mathcal{C}$ corrupted by $\mathcal{A}$ (since it controls the simulater $\mathcal{F}_{\text{DVHCOM}}$) and uses this knowledge to compute new adjusted shares for the simulated honest parties such that the output of this simulated execution is equal to the random outputs obtain from $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$. In the Reveal phase, $\mathcal{S}$ equivocates the opening of the simulated honest party committed shares so that they match the adjusted shares computed before. Verification is simulated by executing the steps of an honest party.

6 Acknowledgements

The authors would like to thank Diego Aranha, Ronald Cramer and Dario Fiore for useful discussions and Eva Palandjian for remarks about the draft.

References

1. Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX Security Symposium*, pages 335–348, 2008.
2. Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Winning the caucus race: Continuous leader election via public randomness. *CoRR*, abs/1801.07965, 2018.
3. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. LNCS, pages 324–356. Springer, Heidelberg, 2017.
4. Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 175–196. Springer, Heidelberg, September 2014.
5. Carsten Baum, Bernardo David, and Rafael Dowsley. A framework for universally composable publicly verifiable cryptographic protocols. Cryptology ePrint Archive, Report 2020/207, 2020. `https://eprint.iacr.org/2020/207`.
6. Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: efficient secure multiparty computation with punishable abort. In *Financial Cryptography*, 2020.
7. G. R. Blakley and Catherine A. Meadows. Security of ramp schemes. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 242–268, 1984.
8. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. LNCS, pages 757–788. Springer, Heidelberg, 2018.
9. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. In *Financial Cryptography and Data Security - FC 2018*, pages 64–77, 2018.
10. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. *IACR Cryptology ePrint Archive*, 2019:1021, 2019.
11. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. LNCS, pages 280–312. Springer, Heidelberg, 2018.
12. Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 449–467. Springer, Heidelberg, December 2011.
13. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
14. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 597–608. ACM Press, November 2014.
15. Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, Rafael Dowsley, and Irene Giacomelli. Efficient UC commitment extension with homomorphism for free (and applications). In *Advances in Cryptology - ASIACRYPT 2019 - Proceedings, Part II*, volume 11922 of *LNCS*, pages 606–635. Springer, 2019.
16. Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, Heidelberg, August 2016.
17. Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In *ACNS 17*, LNCS, pages 537–556. Springer, Heidelberg, 2017.

18. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.
19. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.
20. Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 396–407, 1985.
21. James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.
22. Ronald Cramer and Chen Yuan. Personal communication.
23. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. LNCS, pages 66–98. Springer, Heidelberg, 2018.
24. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.
25. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710, 1992.
26. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.
27. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.
28. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.
29. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.
30. Somayeh Heidarvand and Jorge L. Villar. Public verifiability from pairings in secret sharing schemes. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 294–308. Springer, Heidelberg, August 2009.
31. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. LNCS, pages 357–388. Springer, Heidelberg, 2017.
32. Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.
33. Yehuda Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 93–109. Springer, Heidelberg, March 2015.
34. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

35. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.
36. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.
37. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.
38. Michael O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983.
39. Alexandre Ruiz and Jorge Luis Villar. Publicly verfiable secret sharing from paillier's cryptosystem. In *WEWoRC 2005*, pages 98–108, 2005.
40. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
41. Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy, SP 2017*, pages 444–460, 2017.
42. Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 1367–1383, 2020.
43. Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 137–152, 2015.
44. Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.
45. David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 179–192, 2012.

Appendix

A Cooley-Tukey FFT algorithm

We present the Cooley-Tukey FFT algorithm, as described in [44], in Figure 15.

Cooley-Tukey FFT algorithm

Input: An integer $n = 2^k$ dividing $q - 1$, a primitive n-th root of unity ω, and a polynomial $f = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1}$.

Output: The vector $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$.

Algorithm:
If $n = 1$:
- Return f_0.

Else:
- Compute $g = \sum_{j=0}^{n/2-1} (f_j + f_{j+n/2}) X^j$, $g^* = \sum_{j=0}^{n/2-1} (f_j - f_{j+n/2}) \omega^j X^j$.
- Apply the algorithm recursively on $(n/2, g, \omega^2)$ and on $(n/2, g^*, \omega^2)$ obtaining outputs $(g(1), g(\omega^2), \ldots, g(\omega^{n-2}))$ and $(g^*(1), g^*(\omega^2), \ldots, g^*(\omega^{n-2}))$ respectively.
- Return $(g(1), g^*(1), g(\omega^2), g^*(\omega^2), \ldots, g(\omega^{n-2}), g^*(\omega^{n-2}))$.

Fig. 15. Cooley-Tukey FFT algorithm

B Security proof of PVSS protocol π_{PVSS}

In order to show that π_{PVSS} is IND1-secret under the DDH assumption, we will in fact use the following assumption.

Definition 6 (ℓ-DDH hardness assumption). *Let $\ell := \ell(\lambda)$ be an integer depending on the security parameter λ.*

Let D_0, D_1 be the probability distributions outputting tuples

$$\mathbf{x} := (g, g^{\alpha}, g^{\beta_0}, g^{\beta_1}, \cdots, g^{\beta_{\ell-1}}, g^{\gamma_0}, g^{\gamma_1}, \cdots, g^{\gamma_{\ell-1}})$$

where g is a uniformly random generator of $\mathbb{Z}_q$, $\alpha, \beta_0, \beta_1, \ldots, \beta_{\ell-1}$ are independently sampled uniformly random values in $\mathbb{Z}_q$, and:

- *In the case of D_0, $\gamma_i = \alpha \cdot \beta_i$ for all $i \in [0, \ell - 1]$.*
- *In the case of D_1, $\gamma_0, \gamma_1, \ldots, \gamma_{\ell-1}$ are sampled uniformly at random from $\mathbb{Z}_q$ and independently of each other and of everything else.*

Given a PPT adversary $\mathcal{A}$, we define its distinguishing advantage as the probability

$$Adv(\mathcal{A}) = |Pr[b = b' | b \leftarrow \{0,1\}, \mathbf{x} \leftarrow D_b, b' \leftarrow \mathcal{A}(\mathbf{x})] - 1/2|$$

We say that the ℓ-DDH problem is hard if for all PPT adversary $\mathcal{A}$, $Adv(\mathcal{A})$ is negligible in λ.

Remark 7. Note that the 1-DDH hardness assumption is the usual DDH hardness assumption.

It was shown in [35] that

Proposition 4. *If the DDH-problem is hard, then the ℓ-DDH problem is hard for $\ell(\lambda) = poly(\lambda)$.*

In fact, the reduction is tight and the distinguishing advantage is the same. Now we can prove our result.

Proof (Proof of Proposition3). The proof follows from similar techniques as in the security analysis of the PVSS in SCRAPE [17].

We suppose that an adversary $\mathcal{A}_{\text{Priv}}$ can break the IND1-secrecy property of protocol π_{DDH}, and construct a distinguisher $\mathcal{A}_{\ell-\text{DDH}}$ that uses $\mathcal{A}_{\text{Priv}}$ to break the ℓ-DDH assumption with the same advantage. As pointed out before the ℓ-DDH assumption is hard if the DDH assumption is hard, since $\ell = O(n)$ and n is polynomial in the security parameter. Without loss of generality we can assume $\mathcal{A}_{\text{Priv}}$ corrupts the t first parties.

Let $(g, g^{\alpha}, g^{\beta_0}, g^{\beta_1}, \cdots, g^{\beta_{\ell-1}}, g^{\gamma_0}, g^{\gamma_1}, \cdots, g^{\gamma_{\ell-1}})$ be an instance of the ℓ-DDH problem. Without loss of generality we assume $\alpha \neq 0$, $\beta_i \neq 0$ for all i. $\mathcal{A}_{\ell-\text{DDH}}$ simulates the IND1 game as follows:

1. The challenger sets $h = g^{\alpha}$ as generator for the group. In the Setup phase, for $i \in [t+1, n]$, $\mathcal{A}_{\ell-\text{DDH}}$ chooses uniformly random values $u_i \leftarrow \mathbb{Z}_q$, and defines $pk_i = g^{u_i}$. Implicitely, this defines sk_i as $sk_i = u_i/\alpha$ even though α is not known to $\mathcal{A}_{\ell-\text{DDH}}$. The values pk_i and the generator h are sent to $\mathcal{A}_{\text{Priv}}$.
2. For $i \in [1, t]$, $\mathcal{A}_{\text{Priv}}$ creates sk_i and $pk_i = h^{sk_i}$ and sends this to the challenger.
3. For $i \in [1, t]$, the challenger chooses uniformly random values $\sigma_i \leftarrow \mathbb{Z}_q$ and sets $\hat{\sigma}_i = pk_i^{\sigma_i}$. Let $p(X)$ be the polynomial in $\mathbb{Z}_q[X]_{\leq t+\ell-1}$ such that $p(-j) = \beta_j$ for $j \in [0, \ell-1]$, $p(i) = \sigma_i$ for $i \in [1, t]$. Then $\mathcal{A}_{\ell-\text{DDH}}$ can compute all values $g^{p(i)}$ for $i \in [-(\ell-1), t]$ because for the positive values of i, it knows the exponent and, in the other cases, the values equal g^{β_j} which are part of its input. By Lagrange interpolation applied in the exponent, $\mathcal{A}_{\ell-\text{DDH}}$ can compute $g^{p(i)}$ for $i \in [t+1, n]$. From here, $\mathcal{A}_{\ell-\text{DDH}}$ computes $(g^{p(i)})^{u_i} = h^{\sigma_i}$ for $i \in [t+1, n]$ and sets this as $\hat{\sigma}_i$ for those indices.

To simulate the NIZK proof of knowledge, $\mathcal{A}_{\ell-\mathrm{DDH}}$ chooses a polynomial $z \in \mathbb{Z}_q[X]_{\leq t+\ell-1}$ and $e \in \mathbb{Z}_q$ uniformly at random, sets $a_i = \hat{\sigma}_i^e / pk_i^{z(i)}$ for $i \in [1, n]$ and programs the random oracle $H(\cdot)$ so that

$$e = H(pk_1, \ldots, pk_n, \hat{\sigma}_1, \ldots, \hat{\sigma}_n, a_1, \ldots, a_n).$$

Finally it sends $\hat{\sigma}_i$ for all $i \in [1, n], \sigma_i$ for all $i \in [1, t]$, and $(a_1, ..., a_n, e, z)$ together with the vector $(g^{\gamma_0}, g^{\gamma_1}, \cdots, g^{\gamma_{\ell-1}})$ (which plays the role of x_b in the IND game) to $\mathcal{A}_{\mathrm{Priv}}$.

4. $\mathcal{A}_{\mathrm{Priv}}$ makes a guess b'.

If $b' = 0$, $\mathcal{A}_{\ell-\mathrm{DDH}}$ guesses that $(\gamma_i)_{i\in[0,\ell-1]} = \alpha \cdot (\beta_i)_{i\in[0,\ell-1]}$. If $b' = 1$, $\mathcal{A}_{\ell-\mathrm{DDH}}$ guesses that $(\gamma_0, \gamma_1, \ldots, \gamma_{\ell-1})$ is a random element in $\mathbb{Z}_p^\ell$.

The information that $\mathcal{A}_{\mathrm{Priv}}$ receives in step 3. is distributed exactly like a sharing of the value $h^\beta = g^{\alpha\cdot\beta}$ with the PVSS. Consequently, $\gamma = \alpha \cdot \beta$ if and only if the value g^γ sent to $\mathcal{A}_{\mathrm{Priv}}$ is the secret shared by the PVSS. It is now easy to see that the guessing advantage of $\mathcal{A}_{\ell-\mathrm{DDH}}$ is the same as the advantage of $\mathcal{A}_{\mathrm{Priv}}$.

C Complexity of π_{PPVSS}

In Table 2, we collect the number of exponentiations required by each party in the PVSS. The table shows the numbers for the case when only 1 instance of the PVSS has been used. If the secrets of L instances of the PVSS are reconstructed in parallel, then share decryption can be amortized to $2L+1$ exponentiations per decrypting party and share decryption verification can be amortized to $(2L + 2)(t + \ell)$ exponentiations per verifier, by using the batch DLEQ proof.

Phase	Number of exponentiations	Party
Distribution	$2n + \ell = O(n)$	By dealer only
Verification	$2n = O(n)$	Per verifier
Share Decryption	$3 = O(1)$	Per decrypting party
Share Dec. Verification of $n - t = t + \ell$ shares	$4(t + \ell) = O(n)$	Per verifier
Reconstruction	$\ell(t + \ell) = O(n^2)$	Per reconstructing party
Rec. Verification	$t + 2\ell = O(n)$	Per reconstruction verifier

Table 2. Number of exponentiations for each phase of the PVSS (assuming only 1 instance of the PVSS has been used). Distribution includes the computation of S_j from s_j. In the Share Decryption Verification we count verification of $n - t$ shares, since we need that number to reconstruct the secret.

D Test for Correction for Linear Computation in the Exponent

In case we want to use the alternative distributed computation of the output phase, as mentioned in Section 4.3, we will need an algorithm allowing for local verification of the correct computation of a linear resilient function in the exponent by some other party. We use similar ideas to the algorithm $Local_{LDEI}$. The basis for the algorithm, that we call $Local_{LExp}$ in fact allows for testing the correct computation of an arbitrary deterministic linear function in the exponent given by a matrix M. Consider the set $C = \{(\mathbf{x}, M\mathbf{x}) : \mathbf{x} \in \mathbb{Z}_q^r\} \subseteq \mathbb{Z}_q^{2r-t}$. This is a vector space over $\mathbb{Z}_q$ (i.e. a linear code) of dimension r. Consider now the $(r-t) \times (2r-t)$ matrix

$$H = (-M|I_{r-t})$$

Every row of H is orthogonal to every element in C. Indeed the i-th row of this matrix is $(-M_i|\mathbf{e}_i)$ where M_i is the i-th row of M and $\mathbf{e}_i$ is the i-th unit vector. The inner product between this and $(\mathbf{x}, M\mathbf{x})$ gives $-M_i\mathbf{x} + (M\mathbf{x})_i = -M_i\mathbf{x} + M_i\mathbf{x} = 0$. Moreover the rows of H are all linearly independent and generate a code of dimension $r-t$, which is the dual code $C^\perp$ of C. More precisely $C^\perp = \{(-\mathbf{z}M, \mathbf{z}) : \mathbf{z} \in \mathbb{Z}_q^{r-t}\}$.

Then we have the following standard result (see e.g. [17, Lemma 1]).

Lemma 2. *Let $\mathbf{v} \in \mathbb{Z}_q^{2r-t}$ and let $\mathbf{w} = (-\mathbf{z}M, \mathbf{z})$ uniformly random in $C^\perp$ (i.e. $\mathbf{z}$ is sampled uniformly at random in $\mathbb{Z}_q^{r-t}$). Then:*

- *If $\mathbf{v} \in C$, $\langle \mathbf{v}, \mathbf{w} \rangle = 0$ with probability 1.*
- *If $\mathbf{v} \notin C$, $\langle \mathbf{v}, \mathbf{w} \rangle = 0$ with probability at most $1/q$.*

This gives a probabilistic test of whether a given vector $\mathbf{v} \in \mathbb{Z}_q^{2r-t}$ is in C, i.e., whether it is of the form $(\mathbf{x}, M\mathbf{x})$. In our situation we need to check that given $\mathbf{h} = (h_1, h_2, \ldots, h_r) \in \mathbb{G}_q^r$ and $\widehat{\mathbf{h}} \in \mathbb{G}_q^{r-t}$, it holds that $\widehat{\mathbf{h}} = M \diamond \mathbf{h}$. Recall this means that the discrete logarithms of the elements in $\widehat{\mathbf{h}}$ with respect to some generator h form a vector $\mathbf{y}$ which equals $M\mathbf{x}$, where $\mathbf{x}$ is the vector of discrete logarithms of $\mathbf{h}$ with respect to h. Therefore if we write $(\mathbf{h}, \widehat{\mathbf{h}}) = (h^{z_1}, h^{z_2}, \ldots, h^{z_{2r-t}})$, we are testing whether $\mathbf{z} \in C$. This leads to the testing algorithm $Local_{LExp}$ in Figure 16.

Because of the arguments above, we have

Proposition 5. *If the statement is correct, $Local_{LExp}$ always accepts. If the statement is false, $Local_{LExp}$ accepts with probability at most $1/q$.*

Note that the number of required exponentiations in the group is $2r - t$.

E Universal Composability and Ideal Functionalities

In Sections 5 and 5.2, we work in the Universal Composability (UC) framework of Canetti [13]. In this section, we discuss our treatment of public verifiable functionalities, randomness beacons and setup assumptions in the UC framework. We refer the readers to [13] for further details on this model.

Algorithm $Local_{LExp}$ to Verify Correctness of Linear Function Computation in the Exponent

Public parameters: prime q, cyclic group $\mathbb{G}_q$ of prime order q, integers t, r, matrix $M \in \mathbb{Z}_q^{(r-t)\times r}$.
Input: $\mathbf{h} = (h_1, h_2, \ldots, h_r) \in \mathbb{G}_q^r$ and $\widehat{\mathbf{h}} = (\widehat{h}_1, \widehat{h}_2, \cdots, \widehat{h}_{r-t}) \in \mathbb{G}_q^{r-t}$.
Goal: Verify whether $\widehat{\mathbf{h}} = M \diamond \mathbf{h}$ (Definition 3).
Algorithm:
- Sample $\mathbf{z}$ uniformly at random in $\mathbb{Z}_q^{r-t}$. Let $\mathbf{u} = -\mathbf{z}M$.
- Compute $a = \prod_{i=1}^{r} h_i^{u_i} \cdot \prod_{i=1}^{r-t} \widehat{h}_i^{z_i}$
- If $a = 1$ accept. Otherwise reject.

Fig. 16. Algorithm $Local_{LExp}$ to Verify Correctness of Linear Function Computation in the Exponent

Adversarial Model: Our protocols will be proven secure against static and active adversaries. In other words, the adversary may deviate from the protocol in any arbitrary way and can only corrupt parties before the protocol execution starts.

E.1 Public Verfiability and Dynamic Verifiers

In order to model a dynamic set of verifier parties who may join and leave an execution at any given point, we follow the approach of Badertscher et al. [3], defining a set of verifiers $\mathcal{V}$ to which parties can be added or removed through register and de-register instructions as well as defining instructions that allow $\mathcal{S}$ to obtain the list of registered verifiers. All functionalities that ineract with a set of verifiers $\mathcal{V}$ implicitly include the following interfaces (which are omitted henceforth for simplicity):

Register: Upon receiving (REGISTER, sid) from some verifier $\mathcal{V}_i$, set $\mathcal{V} = \mathcal{V} \cup \mathcal{V}_i$ and return (REGISTERED, $sid, \mathcal{V}_i$) to $\mathcal{V}_i$.

Deregister: Upon receiving (DEREGISTER, sid) from some verifier $\mathcal{V}_i$, set $\mathcal{V} = \mathcal{V} \setminus \mathcal{V}_i$ and return (DEREGISTERED, $sid, \mathcal{V}_i$) to $\mathcal{V}_i$.

Is Registered: Upon receiving (IS-REGISTERED, sid) from $\mathcal{V}_i$, return (IS-REGISTERED, sid, b) to $\mathcal{V}_i$, where $b = 1$ if $\mathcal{V}_i \in \mathcal{V}$ and $b = 0$ otherwise.

Get Registered: Upon receiving (GET-REGISTERED, sid) from the ideal adversary $\mathcal{S}$, the functionality returns (GET-REGISTERED, $sid, \mathcal{V}$) to $\mathcal{S}$.

E.2 Public Bulletin Boards

As in previous works [4,6,32] modelling publicly verifiable protocols in the UC framework, we require a public bulletin board ideal functionality as setup. While

Ideal Functionality $\mathcal{F}_{APBB}$ - Authenticated Public Bulletin Board

Functionality $\mathcal{F}_{APBB}$ keeps an initially empty list $\mathcal{M}$ of messages and interacts with a set of parties $\mathcal{P}$, a set of verifiers $\mathcal{V}$ and an ideal adversary $\mathcal{S}$ as follows:

Post to Bulletin Board: Upon receiving a message $(\text{POST}, sid, \text{MID}, m)$ from a party $\mathcal{P}_i \in \mathcal{P}$, if there is no message $(\mathcal{P}_i, sid, \text{MID}, m') \in \mathcal{M}$, append $(\mathcal{P}_i, sid, \text{MID}, m)$ to the list $\mathcal{M}$ of authenticated messages that were posted in the public bulletin board. Then send $(\text{POSTED}, sid, \mathcal{P}_i, \text{MID}, m)$ to $\mathcal{S}$.

Read from Bulletin Board: Upon receiving a message (READ, sid) from a party in $\mathcal{P}$ or $\mathcal{V}$, return $(\text{READ}, sid, \mathcal{M})$ to the caller.

Fig. 17. Authenticated Public Bulletin Board Ideal Functionality $\mathcal{F}_{APBB}$ from [6].

randomness beacons are usually deployed on top of blockchain based public ledgers [3] using a digital signature ideal functionality to provide authentication, in order to focus on the main aspects of our protocol instead of the many public ledger details, we use the ideally authenticated bulletin board from [6], which is described in Figure 17. However, we remark that our randomness beacon does not depend on the order of the messages posted to the bulletin board and that it could be realized using a blockchain based public ledger using the strategy of [31,23], where parties wait for long enough during each phase (*i.e.* commit, reveal and recover) such that it is guaranteed that all of their messages become immutable and accessible to all honest parties with overwhelming probability.

E.3 Common Reference String (CRS)

For the sake of simplicity, in order to model the public parameters $g, h \in \mathbb{G}_q$ used by the Pedersen equivocal commitment scheme, we use a Common Reference String (CRS) ideal functionality $\mathcal{F}_{\mathsf{CRS}}$. This functionality is also implied by the NIZK ideal functionalities we employ. However, we remark that such public parameters for Pedersen commitments and NIZKs can be computed through a multiparty computation protocol [9] without relying on a trusted CRS. Ideal functionality $\mathcal{F}_{\mathsf{CRS}}$ is described in Figure 18.

Common Reference String ideal functionality $\mathcal{F}_{\mathsf{CRS}}$

Functionality $\mathcal{F}_{\mathsf{CRS}}$ is parameterized by a distribution $\mathcal{D}$ and interacts with a set of parties $\mathcal{P}$ and a set of verifiers $\mathcal{V}$ as follows:

1. On input (CRS, sid) from a party in $\mathcal{P}$ or in $\mathcal{V}$, if there is no value crs recorded, then sample and record value $\mathsf{crs} \xleftarrow{\$} \mathcal{D}$. Send $(\text{CRS}, sid, \mathsf{crs})$ to the caller.

Fig. 18. Common Reference String ideal functionality $\mathcal{F}_{\mathsf{CRS}}$.

E.4 Non-Interactive Zero Knowledge (NIZK)

It is known that a UC-secure NIZKs for all relations in NP can be realized in the CRS model [27,28,29]. However, we remark that more efficient constructions exist for the case of the simple relations on discrete logarithm that we are interested in, more efficient constructions exist [12] and those can be made non-interactive through techniques [33] that can be instantiated based on an observable (*i.e.* non-programmable) global random oracle [14,11] and a CRS. We use the ideal functionality for Non-Interactive Zero Knowledge proof systems for a relation R from [27,28,29], which we describe in Figure 19. For the sake of clarity and keeping with the style of other functionalities, we slightly modify $\mathcal{F}_{\mathsf{NIZK}}^{R}$ and model the set of verifiers $\mathcal{V}$ separately from the set of parties $\mathcal{P}$, only allowing the parties in $\mathcal{P}$ (resp. verifiers in $\mathcal{V}$) to call the proof (resp. verification) interface. However, a party may be added to both sets $\mathcal{P}$ and $\mathcal{V}$ when it needs to query both interfaces.

Non-Interactive Zero Knowledge proof system ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{R}$

Functionality $\mathcal{F}_{\mathsf{NIZK}}^{R}$ is parameterized with relation R and interacts with a set of parties $\mathcal{P}$, a set of verifiers $\mathcal{V}$ and ideal adversary $\mathcal{S}$ as follows:

Proof: On input $(\text{PROVE}, sid, x, w)$ from a party $\mathcal{P}_i \in \mathcal{P}$, ignore if $(x, w) \notin R$. Send (PROVE, sid, x) to $\mathcal{S}$ and wait for answer (PROOF, sid, π). Upon receiving the answer, store (x, π) and send (PROOF, sid, π) to $\mathcal{P}_i$.
Verification: On input $(\text{VERIFY}, sid, x, \pi)$ from a party $\mathcal{V}_i \in \mathcal{V}$, check whether (x, π) is stored. If not, send $(\text{VERIFY}, sid, x, \pi)$ to $\mathcal{S}$ and wait for an answer $(\text{WITNESS}, sid, w)$. Upon receiving the answer, check whether $(x, w) \in R$ and, in that case, store (x, π). If (x, π) has been stored, return $(\text{VERIFICATION}, sid, 1)$ to $\mathcal{V}_i$, else return $(\text{VERIFICATION}, sid, 0)$.

Fig. 19. Non-Interactive Zero Knowledge proof ideal functionality $\mathcal{F}_{\mathsf{NIZK}}^{R}$ from [29].

E.5 Public-Key Encryption with Plaintext Verification:

We present functionality $\mathcal{F}_{\mathsf{PKEPV}}$ from [5] in Figure 20. It has been shown in [5] that $\mathcal{F}_{\mathsf{PKEPV}}$ can be UC-realized in the global random oracle model under the Computational Diffie-Hellman assumption.

E.6 Multi-Receiver Publicly Verifiable Commitments

We use multi-receiver publicly verifiable (non-homomorphic) commitments as the main building block of our multi-receiver additively homomorphic designated verifier commitment scheme. We adopt the multi-receiver publicly verifiable (non-homomorphic) commitments functionality $\mathcal{F}_{\mathsf{Com}}$ from [6], where it is

Functionality $\mathcal{F}_{\mathsf{PKEPV}}$

$\mathcal{F}_{\mathsf{PKEPV}}$ interacts with a special decrypting party $\mathcal{P}_{owner}$, a set of parties $\mathcal{P}$, a set of public verifiers $\mathcal{V}$ and an ideal adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{PKEPV}}$ is parameterized by a message domain ensemble $M = \{M_k\}_{k\in\mathcal{N}}$, a family of formal encryption algorithms $\{E_e\}_e$, a family of formal decryption algorithms $\{D_d\}_d$ for unregistered ciphertexts and a family of formal plaintext verification algorithms $\{V_v\}_v$. $\mathcal{F}_{\mathsf{PKEPV}}$ proceeds as follows:

Key Generation: Upon receiving a message $(\text{KeyGen}, sid, \mathcal{P}_{owner})$ from a party $\mathcal{P}_{owner} \in \mathcal{P}$ (or $\mathcal{S}$), proceed as follows:

1. Send $(\text{KeyGen}, sid, \mathcal{P}_{owner})$ to $\mathcal{S}$.
2. Receive a value $\mathbf{e}$ from $\mathcal{S}$.
3. Record $\mathbf{e}$ and output $\mathbf{e}$ to $\mathcal{P}_{owner}$.

Encryption: Upon receiving a message $(\text{Encrypt}, sid, \mathcal{P}_{owner}, \mathbf{e}', m)$ from a party $\mathcal{P}_i \in \mathcal{P}$, proceed as follows:

1. If $m \notin M$, then return an error message to $\mathcal{P}_i$.
2. If $m \in M$, then:
   - If $\mathcal{P}_{owner}$ is corrupted, or $\mathbf{e}' \neq \mathbf{e}$, then compute $(c, \pi) \leftarrow E_k(m)$.
   - Otherwise, let $(c, \pi) \leftarrow E_k(1^{|m|})$.

   Record the pair (m, c, π) and return (c, π) to $\mathcal{P}_i$.

Decryption: Upon receiving a message $(\text{Decrypt}, sid, \mathcal{P}_{owner}, c)$ from $\mathcal{P}_{owner}$, proceed as follows (if the input is from another party then ignore):

1. If there is a recorded tuple (c, m, π), then hand (m, π) to $\mathcal{P}_{owner}$. (If there is more than one value m that corresponds to c then unique decryption is not possible. In that case, output an error message to $\mathcal{P}_{owner}$).
2. Otherwise, compute $(m, \pi) \leftarrow D(c)$ and hand (m, π) to $\mathcal{P}_{owner}$.

Plaintext Verification: Upon receiving a message $(\text{Verify}, sid, \mathcal{P}_{owner}, c, m, \pi)$ from a verifier $\mathcal{V}_i \in \mathcal{V}$, proceed as follows:

1. If there is a recorded tuple (c, m, π), then output 1 to $\mathcal{V}_i$.
2. Otherwise, compute $b \leftarrow V(c, m, \pi)$, outputting b to $\mathcal{V}_i$.

Fig. 20. Public-Key Encryption Functionality with Plaintext Verification from [5].

also shown that $\mathcal{F}_{\mathsf{Com}}$ can be realized in the restricted programmable and observable random oracle model of [11] with the use of an ideally authenticated public bulletin board $\mathcal{F}_{APBB}$ without extra computational assumptions.

F Realizing $\mathcal{F}_{\text{DVHCOM}}$

We realize $\mathcal{F}_{\text{DVHCOM}}$ with a protocol based on the additively homomorphic multi-receiver commitments from [15] and public verifiability techniques from [5]. The main idea of our protocol is to have the sender of a commitment encrypt the opening information it would usually send to all receivers under the public key of an specific receiver using a public-key cryptosystem with plaintext verification as defined in [5]. Such a cryptosystem, allows the holder of the corresponding public key to publicly prove that the ciphertext contained a certain plaintext. Hence, the receiver of a designated open can later prove to the other receivers that it

Functionality $\mathcal{F}_{\mathsf{Com}}$

$\mathcal{F}_{\mathsf{Com}}$ is parameterized by commitment length λ and keeps an internal (initially empty) list C. $\mathcal{F}_{\mathsf{Com}}$ interacts with a sender P_S, a set of receivers $P = \{P_1, \ldots, P_t\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$, through the following interfaces:

Commit: Upon receiving a message $(\text{COMMIT}, sid, ssid, P_S, P, \boldsymbol{m})$ from P_S where $\boldsymbol{m} \in \{0,1\}^{\lambda}$, check if there exists $(ssid, \cdot, \cdot) \in C$. If yes, ignore the message, else add $(ssid, P_S, \boldsymbol{m})$ to C and send $(\text{RECEIPT}, sid, P_S, P)$ to every receiver $P_i \in P$ and $\mathcal{S}$.

Open: Upon receiving a message $(\text{REVEAL}, sid, ssid)$ from P_S, if there exists $(ssid, P_S, \boldsymbol{m}) \in C$, then send $(\text{REVEAL}, sid, ssid, P_S, P, \boldsymbol{m})$ to every receiver $P_i \in P$ and $\mathcal{S}$.

Verify: Upon receiving $(\text{VERIFY}, sid, ssid, P_S, \boldsymbol{m})$ from $\mathcal{V}_j \in \mathcal{V}$, if there exists $(ssid, P_S, \boldsymbol{m}) \in C$ set $f = 1$, otherwise, set $f = 0$. Send $(\text{VERIFIED}, sid, ssid, P_S, \boldsymbol{m}, f)$ to $\mathcal{V}_j$.

Fig. 21. Functionality $\mathcal{F}_{\mathsf{Com}}$ for Publicly Verifiable Multiparty Commitments from [6].

received a valid designated open or not. In order to achieve public verifiability, we further require an ideal authenticated bulletin board $\mathcal{F}_{APBB}$, where public keys, commitments, public opens and designated opens registered for posterior verification.

F.1 Coding Theory Notation and Tools

We base our protocol on [15] and borrow their notation and tools, which we reproduce almost verbatim here.

Linear Algebra Notation The set of the n first positive integers is denoted $[n] = \{1, 2, \ldots, n\}$. Vectors of elements of some field are denoted by bold lower-case letters, while matrices are denoted by bold upper-case letters. For $\boldsymbol{z} \in \mathbb{F}^k$, $\boldsymbol{z}[i]$ denotes the i'th entry of the vector, where $\boldsymbol{z}[1]$ is the first element of $\boldsymbol{z}$. The coordinate-wise (Schur) product of two vectors is denoted by $*$, *i.e.* if $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$, then $\boldsymbol{a} * \boldsymbol{b} \in \mathbb{F}^n$ and $(\boldsymbol{a} * \boldsymbol{b})[i] = \boldsymbol{a}[i]\boldsymbol{b}[i]$. If $A \subseteq [n]$, we will use π_A to denote the projection that outputs the coordinates with index in A of a vector. For a matrix $\mathbf{M} \in \mathbb{F}^{n \times k}$, we let $\mathbf{M}[\cdot, j]$ denote the j'th column of $\mathbf{M}$ and $\mathbf{M}[i, \cdot]$ denote the i'th row. The row support of $\mathbf{M}$ is the set of indices $I \subseteq \{1, \ldots, n\}$ such that $\mathbf{M}[i, \cdot] \neq \mathbf{0}$. For a vector $\boldsymbol{x} \in \mathbb{F}^n$, we denote the Hamming-weight of $\boldsymbol{x}$ by $\|\boldsymbol{x}\|_0$.

Schur Square and Interleaved Product of Codes The rate of an $\mathbb{F}$-linear $[n, k, d]$ code is $\frac{k}{n}$ and its relative minimum distance is $\frac{d}{n}$. A matrix $\mathbf{G} \in \mathbb{F}^{n \times k}$ is a generator matrix of C if $\mathsf{C} = \{\mathbf{G}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{F}^k\}$, and we write $\mathsf{C}(\boldsymbol{x}) = \mathbf{G}\boldsymbol{x}$. The code C is systematic if it has a generator matrix $\mathbf{G}$ such that the submatrix given by the top k rows of $\mathbf{G}$ is the identity matrix $\mathbf{I} \in \mathbb{F}^{k \times k}$. For an $\mathbb{F}$-linear $[n, k, d]$ code C, we denote by C^{*2} the *Schur square* of C, which is defined as the

linear subspace of $\mathbb{F}^n$ generated by all the possible vectors of the form $\boldsymbol{v} * \boldsymbol{w}$ with $\boldsymbol{v}, \boldsymbol{w} \in \mathsf{C}$. This is an $[n, \hat{k}, \hat{d}]$ code where $\hat{k} \geq k$ and $\hat{d} \leq d$. For an $\mathbb{F}$-linear $[n, k, d]$ code C, we denote by $\mathsf{C}^{\odot m}$ the *m-interleaved product* of C, which is defined by $\mathsf{C}^{\odot m} = \{\mathbf{C} \in \mathbb{F}^{n \times m} : \forall i \in [m] : \mathbf{C}[\cdot, i] \in \mathsf{C}\}$. In other words, $\mathsf{C}^{\odot m}$ consists of all $\mathbb{F}^{n \times m}$ matrices for which all columns are in C. We can think of $\mathsf{C}^{\odot m}$ as a linear code with symbol alphabet $\mathbb{F}^m$, where we obtain codewords by taking m arbitrary codewords of C and bundling together the components of these codewords into symbols from $\mathbb{F}^m$. For a matrix $\mathbf{E} \in \mathbb{F}^{n \times m}$, $\|\mathbf{E}\|_0$ is the number of nonzero rows of $\mathbf{E}$, and the code $\mathsf{C}^{\odot m}$ has minimum distance at least d' if all nonzero $\mathbf{C} \in \mathsf{C}^{\odot m}$ satisfy $\|\mathbf{C}\|_0 \geq d'$. With this definition, it is easy to see that $DistComp(\mathsf{C}^{\odot m}) = DistComp(\mathsf{C})$

Interactive Proximity Testing and Linear Time Building Blocks As in [15], we use the interactive proximity testing technique and corresponding linear time building blocks introduced in [16].

Definition 7 (Almost Universal Linear Hashing [16]). *We say that a family $\mathcal{H}$ of linear functions $\mathbb{F}^n \to \mathbb{F}^s$ is ϵ-almost universal, if it holds for every non-zero $\mathbf{x} \in \mathbb{F}^n$ that*

$$\Pr_{\mathbf{H} \leftarrow \mathcal{H}}[\mathbf{H}(\mathbf{x}) = 0] \leq \epsilon,$$

where $\mathbf{H}$ is chosen uniformly at random from the family $\mathcal{H}$. We say that $\mathcal{H}$ is universal, if it is $|\mathbb{F}^{-s}|$-almost universal. We will identify functions $H \in \mathcal{H}$ with their transformation matrix and write $\mathbf{H}(\mathbf{x}) = \mathbf{H} \cdot \mathbf{x}$.

Theorem 5 (Theorem 1 in [16]). *Let $\mathcal{H} : \mathbb{F}^m \to \mathbb{F}^{2s+t}$ be a family of $|\mathbb{F}|^{-2s}$-almost universal $\mathbb{F}$-linear hash functions. Further let C be an $\mathbb{F}$-linear $[n, k, s]$ code. Then for every $\mathbf{X} \in \mathbb{F}^{n \times m}$ at least one of the following statements holds, except with probability $|\mathbb{F}|^{-s}$ over the choice of $\mathbf{H} \leftarrow \mathcal{H}$:*

1. *$\mathbf{X}\mathbf{H}^\top$ has distance at least s from $\mathsf{C}^{\odot(2s+t)}$*
2. *For every $\mathbf{C}' \in \mathsf{C}^{\odot(2s+t)}$ there exists a $\mathbf{C} \in \mathsf{C}^{\odot m}$ such that $\mathbf{X}\mathbf{H}^\top - \mathbf{C}'$ and $\mathbf{X} - \mathbf{C}$ have the same row support*

Remark 8 ([16]). If the first item in the statement of the Theorem does not hold, the second one must hold. Then we can efficiently recover a codeword $\mathbf{C}$ with distance at most $s - 1$ from $\mathbf{X}$ using erasure correction, given a codeword $\mathbf{C}' \in \mathsf{C}^{\odot(2s+t)}$ with distance at most $s - 1$ from $\mathbf{X}\mathbf{H}^\top$. More specifically, we compute the row support of $\mathbf{X}\mathbf{H}^\top - \mathbf{C}'$, erase the corresponding rows of $\mathbf{X}$ and recover $\mathbf{C}$ from $\mathbf{X}$ using erasure correction (recall that erasure correction for linear codes can be performed efficiently via gaussian elimination). The last step is possible as the distance between $\mathbf{X}$ and $\mathbf{C}$ is at most $s - 1$.

Theorem 6 (Theorem 2 in [15]). *Fix a finite field $\mathbb{F}$ of constant size, let $s \in \mathbb{N}$ be a statistical security parameter, let $n \in \mathbb{N}$ and let $l = s + O(\log(n))$. Then there exists an explicit family $\mathcal{H} : \mathbb{F}^{l+n} \to \mathbb{F}^l$ of $|\mathbb{F}|^{-s}$-universal hash functions that can be represented by $O(s^2)$ bits and computed in time $O(n)$. Moreover,*

it holds for any function $H \in \mathcal{H}$ that if $\mathbf{x} = (x_1, \ldots, x_l, \ldots x_{l+n})$ is such that the $x_1, \ldots, x_l$ are independently uniform and $x_{l+1}, \ldots, x_{l+n}$ are independent of $x_1, \ldots, x_l$, then $H(\mathbf{x})$ is distributed uniformly random.

F.2 Protocol Π_{DVHCOM}

We construct Protocol Π_{DVHCOM} realizing $\mathcal{F}_{\text{DVHCOM}}$ in the $\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{APBB}$-hybrid model by adapting the opening phase of the additively homomorphic commitment scheme of [15]. For the sake of completeness, we reproduce the Commitment and Addition phases of [15] in Figure 22 almost verbatim. The main novelty in our protocol lies in the opening phase, where we realize both regular public openings and designated openings with the help of a public-key encryption with plaintext verification functionality $\mathcal{F}_{\mathsf{PKEPV}}$. Moreover, we show how to perform public verification of openings that have been revealed publicly (either by the sender or by a designated verifier). The new steps for designated openings, public openings, public verification and other auxiliary procedures of Π_{DVHCOM} are described in Figures 23 and 24.

The use of $\mathcal{F}_{APBB}$ instead of broadcast The sole difference in our protocol is that messages are broadcast through $\mathcal{F}_{APBB}$ instead of a regular broadcast channel. In Figure 22, all "broadcast message m" should be read as "send (POST, sid, MID, m) to $\mathcal{F}_{APBB}$ using a fresh MID". Moreover, at every point that parties are expected to receive broadcasts, they send (READ, sid) to $\mathcal{F}_{APBB}$, receive (READ, $sid, \mathcal{M}$) and check that the message they are waiting for is in $\mathcal{M}$.

Public Verification of $\mathcal{F}_{\mathsf{Com}}$ commitments In order to realize the public verification interface of $\mathcal{F}_{\text{DVHCOM}}$, we need to also verify the commitments handled in Π_{DVHCOM} by the underlying instances of $\mathcal{F}_{\mathsf{Com}}$. Since only the sender and the receivers participate in the opening of a $\mathcal{F}_{\mathsf{Com}}$ commitment, we implicitly assume that the sender sends the opened message $(ssid, m)$ to $\mathcal{F}_{APBB}$ so that later on verifiers who do not participate in Π_{DVHCOM} execution can verify that these commitments were opened correctly with the help of $\mathcal{F}_{\mathsf{Com}}$ and the opened values in $\mathcal{F}_{APBB}$.

Using Fiat-Shamir to make Commitment/Open non-interactive As observed in [15], the Fiat-Shamir transformation can be used to make Protocol Π_{DVHCOM} completely non-interactive. The basic idea is to derive the randomness generated by the set of receivers P through commit-then-reveal coin-tossing using a random oracle instead (*i.e.* using the output of the random oracle when queried on the current protocol transcript). Hence, the sender is able to both commit and then open specific commitments without interacting with the receivers.

Commitment to Arbitrary Messages Protocol Π_{DVHCOM} actually realizes a version of $\mathcal{F}_{\text{DVHCOM}}$ that only supports commitments to random messages. However, it has been proven in [15] that this is sufficient to realize commitments to

arbitrary messages. The idea is simple: use the random messages as one-time pad keys to encrypt the arbitrary messages. Since these one-time pads are linear, the linear functions evaluated on the random messages can be replicated by evaluating the same functions on the one-time pad ciphertexts containing the arbitrary messages and decrypting with the result of the linear function evaluated on the random messages as key.

Theorem 7. *Protocol Π_{DVHCOM} UC-realizes $\mathcal{F}_{\text{DVHCOM}}$ in the $\mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{PKEPV}}$-hybrid model with static security against an active adversary $\mathcal{A}$ corrupting all but one party (i.e. corrupting $P_S \bigcup P \setminus P_h$ or corrupting P but not P_S).*

Proof. The proof for the basic Commitment, Addition and (Public) Open phases follows from the analysis of [15], which constructed the original additively homomorphic multi-receiver commitment protocol. The public verifiability by leveraging a verifiable non-homomorphic commitment scheme and an ideally authenticated bulletin board follows from the analysis of [6,15], which showed how to use a similar construction to add public verifiability to the basic scheme of [15]. What is left to analyze is the designated open and reveal of designated opened commitments.

Notice that the designated open is simply constructed by encrypting the original opening information of the protocol in [15] with the help of $\mathcal{F}_{\mathsf{PKEPV}}$ and posting the resulting ciphertext on $\mathcal{F}_{APBB}$. In case the sender P_S is corrupted, by extracting the plaintext commitment opening information from this ciphertext, the simulator can determine whether it contains a valid designated opening and instruct $\mathcal{F}_{\text{DVHCOM}}$ to perform the same designated opening. In case only receivers are corrupted, the simulator simply encrypts the equivocated commitment opening containing the message received from $\mathcal{F}_{\text{DVHCOM}}$ in order to simulate this step. Later on, simulating the reveal of a designated open is achieved by following the instructions of a honest party. Since $\mathcal{F}_{\mathsf{PKEPV}}$ itself guarantees that decryptions can be publicly verified, this designated opening procedure does not affect public verifiability.

Proposition 6. *Protocol Π_{DVHCOM} UC-realizes $\mathcal{F}_{\text{DVHCOM}}$ in the $\mathcal{F}_{GRO}\mathcal{F}_{APBB}$-hybrid model (i.e. in the global random oracle model with an ideally authenticated bulletin board) under the CDH assumption.*

Proof. As proven in [5], a cryptosystem with plaintext verification $\mathcal{F}_{\mathsf{PKEPV}}$ can be realized from a global random oracle under the Computational Diffie-Hellman (CDH) assumption. Using the result from [11] showing that non-homomorphic UC commitments $\mathcal{F}_{\mathsf{Com}}$ can be realized solely from a global random oracle, we arrive at the result that Π_{DVHCOM} realizes $\mathcal{F}_{\text{DVHCOM}}$ under the CDH assumption in the global random oracle model given an authenticated public bulletin board $\mathcal{F}_{APBB}$.

G Security Analysis of Protocol Π_{CT-ZK} - Proof of Theorem 3

We construct a simulator $\mathcal{S}$ such that no environment can distinguish an ideal execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ from a real execution with an adversary $\mathcal{A}$ corrupting up to t parties in the $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{NIZK}}^{LDEI}, \mathcal{F}_{\mathsf{NIZK}}^{DLEQ}, \mathcal{F}_{\mathsf{NIZK}}^{COMC}$-hybrid model. $\mathcal{S}$ simulates an execution of Π_{CT-ZK} with an internal copy $\mathcal{A}$ of the adversary, acting as $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{NIZK}}^{LDEI}, \mathcal{F}_{\mathsf{NIZK}}^{DLEQ}, \mathcal{F}_{\mathsf{NIZK}}^{COMC}$ towards $\mathcal{A}$ by following the exact descriptions of these functionalities unless otherwise stated. One crucial difference is that $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{CRS}}$ towards $\mathcal{A}$ by sampling matrix M correctly but sampling a random generator $g_p \xleftarrow{\$} \mathbb{G}_q$ and computing $h_p = g^{\mathsf{td}}$ for $\mathsf{td} \xleftarrow{\$} \mathbb{Z}_q$, so that it knows a trapdoor td that allows it to equivocate openings of the Pedersen commitment. The case were all parties are honest is handled trivially by executing the exact instructions of Π_{CT-ZK} among simulated honest parties, equivocating the openings of one party's commitment so that the k outputs match those obtained from $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ and outputting whatever $\mathcal{A}$ outputs.

In order to deal with the case where $\mathcal{A}$ corrupts at most t parties, $\mathcal{S}$ executes the Commit phase with $\mathcal{A}$ by following the exact steps of an honest party in the Commit phase of Π_{CT-ZK}. In the Reveal phase, once $n-t$ parties (we say those parties are in set $\mathcal{C}$) post values $\hat{\sigma}_1^j, \ldots, \hat{\sigma}_n^j$, π_{LDEI}^j valid according to $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ and values $\mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j), \pi_{COMC}^j$ valid according to $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$ have been posted on $\mathcal{F}_{APBB}$, $\sim$ extracts from $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$ and $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ the secrets $s_0^j, \ldots, s_{\ell-1}^j$ used as witness for these NIZKs. Notice that at this point, since these values are valid according to $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ and $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$, they are guaranteed to contain the same secrets, meaning that the same secrets will be recovered if all parties in $\mathcal{C}$ open their commitments or if their random inputs are reconstructed from the simulated honest parties' encrypted shares. $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ with (Toss, sid), obtaining (Tossed, $sid, x_1, \ldots, x_k$). $\mathcal{S}$ samples random secrets $\hat{s}_0^j, \ldots, \hat{s}_{\ell-1}^j$ for every simulated honest party $\mathcal{P}_j \in \mathcal{C}$ such that executing the remainder of Π_{CT-ZK} following the instructions of an honest party will yield the output $x_1, \ldots, x_k$ obtained from $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ given that the corrupted parties in $\mathcal{C}$ have the secrets extracted from $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ and $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$. In the remainder of the protocol, $\sim$ executes exactly the steps of an honest party in Π_{CT-ZK} except for equivocating the opening of the commitments $\mathsf{Com}(s_0^j, r_0^j), \ldots, \mathsf{Com}(s_{\ell-1}^j, r_{\ell-1}^j)$ of simulated honest parties in $\mathcal{C}$ so that they open to the new adjusted $\hat{s}_0^j, \ldots, \hat{s}_{\ell-1}^j$ instead of the original random secrets (which $\mathcal{S}$ can do since it knows td).

In order to argue why this simulated execution is indistinguishable from a real execution of Π_{CT-ZK}, we first observe that under the DDH assumption the encrypted shares do not reveal any information about the random inputs shared by the parties as per Proposition 3. Moreover, notice that the Pedersen commitments computed as part of Π_{CT-ZK} are perfectly hiding, also not revealing any information about the shared random inputs. By the time the $n-t$ parties in set $\mathcal{C}$ post their valid encrypted shares and commitments to secrets, we are guaranteed by $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ and $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$ that the encrypted shares and commitments encode the same secrets. Under the discrete logarithm assumption (which is of

course implied by DDH), we also know that $\mathcal{A}$ cannot open the commitments of corrupted parties in $\mathcal{C}$ to any other values than the secret extracted from $\mathcal{F}_{\mathsf{NIZK}}^{LDEI}$ and $\mathcal{F}_{\mathsf{NIZK}}^{COMC}$. Hence, we know that whether $\mathcal{A}$ opens its commitments or we simulate a reconstruction of $\mathcal{A}$'s secrets, the obtained secrets will always be the ones $\mathcal{S}$ has extracted from the NIZK functionalities. Now $\mathcal{S}$ can obtain the output output $x_1, \ldots, x_k$ from $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ and sample new secrets for the simulated honest parties such that executing the remainder of Π_{CT-ZK} with $\mathcal{A}$ will yield the output of $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$. Notice that these new adjusted secrets are sampled at random and thus distributed exactly as in a real execution of Π_{CT-ZK}. Moreover, since $\mathcal{S}$ knows the trapdoor td for the Pedersen equivocal commitments it can open them to these new adjusted values with openings that are distributed exactly as in a real execution of Π_{CT-ZK}. Hence, the ideal execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CT}}^{k,\mathcal{D}}$ and the real execution with $\mathcal{A}$ in the $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{NIZK}}^{LDEI}, \mathcal{F}_{\mathsf{NIZK}}^{DLEQ}, \mathcal{F}_{\mathsf{NIZK}}^{COMC}$-hybrid model are indistringuishable to the environment.

Protocol Π_{DVHCOM}

Let C be a systematic binary linear $[n, k, s]$ code, where s is the statistical security parameter and n is $k + O(s)$. Let $\mathcal{H}$ be a family of linear almost universal hash functions $\mathbf{H} : \{0,1\}^m \to \{0,1\}^l$. Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{m+l}$ be a pseudorandom generator. Protocol Π_{DVHCOM} is run by a sender P_S and a set of receivers $P = \{P_1, \ldots, P_t\}$, who interact with $\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{APBB}, \mathcal{F}_{\mathsf{PKEPV}}$ and proceed as follows:

Commitment Phase

1. On input $(\mathsf{commit}, sid, ssid_1, \ldots, ssid_m, P_S, P)$, P_S proceeds as follows:
   (a) For $i \in [n]$ and $j \in \{0,1\}$, sample $\boldsymbol{s}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$ and send $(\mathsf{commit}, sid, ssid_{i,j}, P_S, P, \boldsymbol{s}_{i,j})$ to $\mathcal{F}_{\mathsf{Com}}$.
   (b) Compute $\mathbf{R_j}[i, \cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,j})$ and set $\mathbf{R} = \mathbf{R_0} + \mathbf{R_1}$ so that $\mathbf{R_0}, \mathbf{R_1}$ forms an additive secret sharing of $\mathbf{R}$.Adjust the bottom $n - k$ rows of $\mathbf{R}$ so that all columns are codewords in C by constructing a matrix $\mathbf{W}$ with dimensions as $\mathbf{R}$ and 0s in the top k rows, such that $\mathbf{A} := \mathbf{R} + \mathbf{W} \in \mathsf{C}^{\odot m+l}$ (recall that C is systematic). Set $\mathbf{A_0} = \mathbf{R_0}, \mathbf{A_1} = \mathbf{R_1} + \mathbf{W}$ and broadcast $(sid, ssid_1, \ldots, ssid_m, \mathbf{W})$ (only sending the bottom $n - k = O(s)$ rows).
2. Upon receiving all messages $(\mathsf{receipt}, sid, ssid_{i,j}, P_S, P)$ from $\mathcal{F}_{\mathsf{Com}}$ and $(sid, ssid_1, \ldots, ssid_m, \mathbf{W})$ from P_S, every receiver $P_i \in P$ proceeds as follows:
   (a) Sample $\boldsymbol{r}_i \xleftarrow{\$} \{0,1\}^n$ and $\boldsymbol{r}'_i \xleftarrow{\$} \{0,1\}^\lambda$, and send $(\mathsf{commit}, sid, ssid, P_i, P', \boldsymbol{r_i})$ and $(\mathsf{commit}, sid, ssid', P_i, P', \boldsymbol{r_i}')$ to $\mathcal{F}_{\mathsf{Com}}$[a], where $P' = P_S \cup P \setminus P_i$.
   (b) Upon receiving $(\mathsf{receipt}, sid, ssid, P_j, P')$ and $(\mathsf{receipt}, sid, ssid', P_j, P')$ from $\mathcal{F}_{\mathsf{Com}}$ for all $P_j \in P \setminus P_i$, send $(\mathsf{reveal}, sid, ssid')$ to $\mathcal{F}_{\mathsf{Com}}$. Upon receiving $(\mathsf{reveal}, sid, ssid', P_j, P', \boldsymbol{r_j}')$ from $\mathcal{F}_{\mathsf{Com}}$ for all $P_j \in P \setminus P_i$, set $\boldsymbol{r}' = \boldsymbol{r_1}' \oplus \ldots \oplus \boldsymbol{r_t}'$.
3. Upon receiving $(\mathsf{commit}, sid, ssid, P_i, P')$ and $(\mathsf{reveal}, sid, ssid', P_j, P', \boldsymbol{r_j}')$ from $\mathcal{F}_{\mathsf{Com}}$ for all $P_j \in P$, P_S proceeds as follows:
   (a) Use $\boldsymbol{r}' = \boldsymbol{r_1}' \oplus \ldots \oplus \boldsymbol{r_t}'$ as a seed for a random function $\mathbf{H} \in \mathcal{H}$ (note that we identify the function with its matrix and all functions in $\mathcal{H}$ are linear).
   (b) Set matrices $\mathbf{P}$, $\mathbf{P_0}$ and $\mathbf{P_1}$ as the first l columns of $\mathbf{A}$, $\mathbf{A_0}$ and $\mathbf{A_1}$, respectively, and remove these columns from $\mathbf{A}$, $\mathbf{A_0}$ and $\mathbf{A_1}$. Renumber the remaining columns of $\mathbf{A}$, $\mathbf{A_0}$ and $\mathbf{A_1}$ from 1 and associate each $ssid_i$ with the corresponding column index. Notice that $\mathbf{P} = \mathbf{P_0} + \mathbf{P_1}$. For $i \in \{0,1\}$, compute $\mathbf{T}_i = \mathbf{A}_i\mathbf{H} + \mathbf{P}_i$ and broadcast $(sid, ssid_1, \ldots, ssid_m, \mathbf{T_0}, \mathbf{T_1})$. Note that $\mathbf{AH} + \mathbf{P} = \mathbf{A_0H} + \mathbf{P_0} + \mathbf{A_1H} + \mathbf{P_1} = \mathbf{T_0} + \mathbf{T_1}$, and $\mathbf{AH} + \mathbf{P} \in \mathsf{C}^{\odot l}$.

Addition of Commitments

1. On input $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_S, P)$, P_S finds indexes i and j corresponding to $ssid_1$ and $ssid_2$ respectively and check that $ssid_3$ is unused. P_S appends the column $\mathbf{A}[\cdot, i] + \mathbf{A}[\cdot, j]$ to $\mathbf{A}$, likewise appends to $\mathbf{A_0}$ and $\mathbf{A_1}$ the sum of their i-th and j-th columns, and associates $ssid_3$ with the new column index. P_S broadcasts $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$. Note that this maintains the properties $\mathbf{A} = \mathbf{A_0} + \mathbf{A_1}$ and $\mathbf{A} \in \mathsf{C}^{\odot m'}$, where m' is the current number of columns (after appending columns for addition results).
2. Upon receiving $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$, every $P_i \in P$ stores the message.

[a] We assume that each receiver P_i in Π_{DVHCOM} has access to an instance of $\mathcal{F}_{\mathsf{Com}}$ where it acts as sender and where all receivers plus sender P_S act as receivers.

Fig. 22. Commit and Addition phases for the protocol Π_{DVHCOM} from [15].

Protocol Π_{DVHCOM}

Schedule Public Open: On input $(\mathsf{P}-\mathsf{Open}, sid, ssid)$ P_S appends $ssid$ to an initially empty list $\mathsf{open}_{\mathsf{pub}}$.

Schedule Designated Open: On input $(\mathsf{D}-\mathsf{Open}, sid, P_d, ssid)$, P_S appends $(P_d, ssid)$ to an initially empty list $\mathsf{open}_{\mathsf{des}}$.

Execute Open: On input $(\mathsf{do}-\mathsf{Open}, sid)$ P_S proceeds as follows:

1. (a) For every pair $(P_d, ssid) \in \mathsf{open}_{\mathsf{des}}$, P_S finds the index j associated to $ssid$ and sends $(\text{ENCRYPT}, sid, P_d, \mathbf{e}_d, (ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j]))$ to the instance $\mathcal{F}_{\mathsf{PKEPV}}$ for which P_d acts as $\mathcal{P}_{owner}$, receiving (c, π) as response. P_S sends $(\text{POST}, sid, \text{MID}, (P_d, ssid, c))$ to $\mathcal{F}_{APBB}$ using a fresh MID.
   (b) P_S finds the set $J_{Pub} = \{j_1, \ldots, j_o\}$ of indexes associated to every $ssid \in \mathsf{open}_{\mathsf{pub}}$ and sends $(\text{POST}, sid, \text{MID}, (\{ssid \in \mathsf{open}_{\mathsf{pub}}\}, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j])_{j \in J_{Pub}}))$ to $\mathcal{F}_{APBB}$ using a fresh MID.
2. Upon receiving message $(P_S, sid, \text{MID}, (\{ssid \in \mathsf{open}_{\mathsf{pub}}\}, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j])_{j \in J_{Pub}}))$ through $\mathcal{F}_{APBB}$, every $P_i \in P$ sends $(\mathsf{reveal}, sid, ssid)$ to $\mathcal{F}_{\mathsf{Com}}$ and waits for $(\mathsf{reveal}, sid, ssid, P_j, P', \boldsymbol{r_j})$ from $\mathcal{F}_{\mathsf{Com}}$ for all $P_j \in P \setminus P_i$. P_i sets $\boldsymbol{r} = \boldsymbol{r_1} \oplus \cdots \oplus \boldsymbol{r_t}$ and sets the diagonal matrix $\boldsymbol{\Delta}$ such that it contains $\boldsymbol{r}[1], \ldots, \boldsymbol{r}[n]$ in the diagonal.
3. Upon receiving $(\mathsf{reveal}, sid, ssid, P_j, P', \boldsymbol{r_j})$ from $\mathcal{F}_{\mathsf{Com}}$ for all $P_j \in P$, P_S sets $\boldsymbol{r} = \boldsymbol{r_1} \oplus \ldots \oplus \boldsymbol{r_t}$, sends $(\mathsf{reveal}, sid, ssid_{i,\boldsymbol{r}[i]})$ to $\mathcal{F}_{\mathsf{Com}}$ for $i \in [n]$ and halts.
4. Upon receiving $(\mathsf{reveal}, sid, ssid_{i,\boldsymbol{r}[i]}, P_S, P, \boldsymbol{s}_{i,\boldsymbol{r}[i]})$ from $\mathcal{F}_{\mathsf{Com}}$ for $i \in [n]$, every receiver $P_j \in P$ proceeds as folllows:
   (a) Compute $\mathbf{S}[i, \cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,\boldsymbol{r}[i]})$, obtaining a matrix $\mathbf{S}$. Note that each row of $\mathbf{S}$ is a row from either $\mathbf{R_0}$ or $\mathbf{R_1}$, which form an additive secret sharing of $\mathbf{R}$ held by P_S. Set $\mathbf{B} = \boldsymbol{\Delta}\mathbf{W} + \mathbf{S}$. Define the matrix $\mathbf{Q}$ as the first l columns of $\mathbf{B}$ and remove these columns from $\mathbf{B}$, renumbering the remaining columns from 1. Note that, for $\mathbf{A}$ from the commitment phase, $\mathbf{A} = \mathbf{A_0} + \mathbf{A_1}$, $\mathbf{B} = \boldsymbol{\Delta}\mathbf{A_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{A_0}$, $\mathbf{A} \in \mathsf{C}^{\odot m}$, *i.e.*, $\mathbf{A}$ initially held by P_S is additively shared and for each row index, P knows either a row from $\mathbf{A_0}$ or from $\mathbf{A_1}$. Check that $\boldsymbol{\Delta}\mathbf{T_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{T_0} = \mathbf{BH} + \mathbf{Q}$ and that $\mathbf{T_0} + \mathbf{T_1} \in \mathsf{C}^{\odot l}$. If any check fails, abort. Notice that $\mathbf{T_0}, \mathbf{T_1}$ form an additive sharing of $\mathbf{AH} + \mathbf{P}$, where P knows some of the shares, namely the rows of $\mathbf{BH} + \mathbf{Q}$. For every message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$ received from P_S, append $\mathbf{B}[\cdot, j] + \mathbf{B}[\cdot, i]$ to $\mathbf{B}$, where i and j are the index corresponding to $ssid_1$ and $ssid_2$ respectively and associate $ssid_3$ with the new column index. Note that this maintains the property $\mathbf{B} = \boldsymbol{\Delta}\mathbf{A_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{A_0}$.
   (b) For every message $(P_S, sid, \text{MID}, (P_j, ssid, c))$ received through $\mathcal{F}_{APBB}$, P_j sends $(\text{DECRYPT}, sid, \mathcal{P}_{owner}, c)$ to the instance of $\mathcal{F}_{\mathsf{PKEPV}}$ for which it acts as owner, receiving $((ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j]), \pi_{ssid})$ and adding j to an initially empty set J_i (the set of indexes j of commitments for which P_i is a designated verifier).
   (c) For every $j \in J_{Pub} \bigcup J_i$, check that $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j] \in \mathsf{C}$ and that, for $i \in [n]$, it holds that $\mathbf{B}[i, j] = \mathbf{A}_{\boldsymbol{r}[i]}[i, j]$ (recall that $\boldsymbol{r}[i]$ is the i-th entry on the diagonal of $\boldsymbol{\Delta}$). If all checks succeed, for every $j \in J_{Pub} \bigcup J_i$, output the first k positions in $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j]$ as the opened string. Otherwise, abort by outputting $(sid, ssid_j, \perp)$.

Fig. 23. Opening phase for the protocol Π_{DVHCOM}.

Protocol Π_{DVHCOM}

Public Key Registration: When activated for the first time every party $\mathcal{P}_i \in P_S \bigcup P$ sends (KEYGEN, $sid, \mathcal{P}_i$) to an instance of $\mathcal{F}_{\mathsf{PKEPV}}$ towards which it acts as $\mathcal{P}_{owner}$, receiving a public key $\mathbf{e}_i$ and sending (POST, sid, MID, $\mathbf{e}_j$) to $\mathcal{F}_{APBB}$ using a fresh MID. $\mathcal{P}_i$ sends (READ, sid) to $\mathcal{F}_{APBB}$, receives (READ, $sid, \mathcal{M}$) and checks that there exists $(\mathcal{P}_j, sid, \text{MID}, \mathbf{e}_j) \in \mathcal{M}$ for every $\mathcal{P}_j \in P_S \bigcup P \setminus \mathcal{P}_i$.

Reveal Designated Open

1. Upon receiving message (REVEAL-D-OPEN, $sid, P_d, ssid$), if it received a message $(P_S, sid, \text{MID}, (P_j, ssid, c))$ through $\mathcal{F}_{APBB}$, P_d sends sends (POST, sid, MID, $(ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j], \pi_{ssid})$) to $\mathcal{F}_{APBB}$ using a fresh MID.
2. Upon receiving a message $(P_d, sid, \text{MID}, ((ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j]), \pi_{ssid}))$ through $\mathcal{F}_{APBB}$, every receiver $P_i \in P \setminus P_d$ proceeds as follows:
   (a) Retrieve message $(P_S, sid, \text{MID}, (P_d, ssid, c))$ received through $\mathcal{F}_{APBB}$ and send (VERIFY, $sid, P_d, c, ((ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j]), \pi_{ssid})$ to the instance of $\mathcal{F}_{\mathsf{PKEPV}}$ for which P_d acts as $\mathcal{P}_{owner}$, receiving b as response and checking that $b = 1$.
   (b) Check that $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j] \in \mathsf{C}$ and that, for $i \in [n]$, it holds that $\mathbf{B}[i, j] = \mathbf{A}_{\boldsymbol{r}[\mathbf{i}]}[i, j]$.
   (c) If all checks succeed set $\boldsymbol{m}$ to the first k positions in $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j]$. Otherwise, set $\boldsymbol{m} = \perp$. Output (P-REVEAL, $sid, P_S, P, ssid, \boldsymbol{m}$).

Verify: On input (VERIFY, $sid, ssid, P_S, \boldsymbol{m}$), a verifier $\mathcal{V}_i \in \mathcal{V}$ proceeds as follows:

1. Send (READ, sid) to $\mathcal{F}_{APBB}$, receiving (READ, $sid, \mathcal{M}$).
2. Check that every opening of a $\mathcal{F}_{\mathsf{Com}}$ commitment registered in $\mathcal{M}$ is valid according to $\mathcal{F}_{\mathsf{Com}}$. If this check fails, Output (VERIFIED, $sid, ssid, P_S, \boldsymbol{m}, 0$) and ignore the next steps.
3. Execute all the steps of the Commitment and Addition phases following the instructions of a honest receiver P_i and using the messages and commitment openings in $\mathcal{M}$. Execute the steps of the Execute Open phase up to Step 4(c) following the instructions of an honest receiver P_i. If an honest receiver would have aborted, output (VERIFIED, $sid, ssid, P_S, \boldsymbol{m}, 0$) and ignore next steps.
4. If there exists a message $(P_S, sid, \text{MID}, (\{ssid \in \mathsf{open}_{\mathsf{pub}}\}, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j])_{j \in J_{Pub}})) \in \mathcal{M}$ such that $ssid \in \{ssid \in \mathsf{open}_{\mathsf{pub}}\}$ (*i.e.* the commitment identified by $ssid$ was publicly opened), for the index j associated to $ssid$, check that $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j] \in \mathsf{C}$ and that, for $i \in [n]$, it holds that $\mathbf{B}[i, j] = \mathbf{A}_{\boldsymbol{r}[\mathbf{i}]}[i, j]$. If this check passes, set $\boldsymbol{m}'$ to the first k positions in $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j]$ and check that $\boldsymbol{m} = \boldsymbol{m}'$. If all of these checks succeed, set $f = 1$, else, set $f = 0$. Output (VERIFIED, $sid, ssid, P_S, \boldsymbol{m}, f$).
5. If there exists a message $(P_S, sid, \text{MID}, (P_d, ssid, c)) \in \mathcal{M}$ (*i.e. the commitment identified by ssid was opened towards a designated verifier* P_d), check that there exists a message $(P_d, sid, \text{MID}, ((ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j]), \pi_{ssid})) \in \mathcal{M}$ such that $(ssid, \mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j])$ is a valid decryption of c according to the instance of $\mathcal{F}_{\mathsf{PKEPV}}$ for which P_d acts as $\mathcal{P}_{owner}$ (*i.e.* the designated verifier P_d revealed a valid opening). check that $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j] \in \mathsf{C}$ and that, for $i \in [n]$, it holds that $\mathbf{B}[i, j] = \mathbf{A}_{\boldsymbol{r}[\mathbf{i}]}[i, j]$. If this check passes, set $\boldsymbol{m}'$ to the first k positions in $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j]$ and check that $\boldsymbol{m} = \boldsymbol{m}'$. If all of these checks succeed, set $f = 1$, else, set $f = 0$. Output (VERIFIED, $sid, ssid, P_S, \boldsymbol{m}, f$).

Fig. 24. Designated Open Release and Verification phases for the protocol Π_{DVHCOM}.