A Non-Interactive Shuffle Argument With Low Trust Assumptions

Antonis Aggelakis[1], Prastudy Fauzi[2], Georgios Korfiatis[1], Panos Louridas[1], Foteinos Mergoupis-Anagnou[1], Janno Siim[3], and Michał Zając[4]

[1] Greek Research and Technology Network, Athens, Greece
[2] Simula UiB, Bergen, Norway
[3] University of Tartu, Tartu, Estonia
[4] Clearmatics, London, UK

Abstract. A shuffle argument is a cryptographic primitive for proving correct behaviour of mix-networks without leaking any private information. Several recent constructions of non-interactive shuffle arguments avoid the random oracle model but require the public key to be trusted.

We augment the most efficient argument by Fauzi et al. [Asiacrypt 2017] with a distributed key generation protocol that assures soundness of the argument if at least one party in the protocol is honest and additionally provide a key verification algorithm which guarantees zero-knowledge even if all the parties are malicious. Furthermore, we simplify their construction and improve security by using weaker assumptions while retaining roughly the same level of efficiency. We also provide an implementation to the distributed key generation protocol and the shuffle argument.

Keywords: subversion security, non-interactive zero-knowledge, shuffle, secure multi-party computation

1 Introduction

Due to convenience for voters and lower election costs, internet voting (i-voting) is becoming an increasingly popular alternative to paper-based voting. In fact, some countries have already provided i-voting solutions in regional (e.g., Australia, Switzerland) or even national (e.g., Estonia) elections. While i-voting has many benefits, the opposing requirements of election transparency and voter's privacy are not easy to guarantee in the digital setting.

One common tool to improve voter's privacy is the mix-network [Cha81]. Essentially, a mix-network can be seen as a digital analogue to ballot-box shaking in paper-based voting. During the voting phase, encrypted votes are sent to a *bulletin board*, a secure append-only storage system. After the voting phase ends, the ciphertexts are processed sequentially by a mix-network consisting of multiple independent servers, called mixers. Each mixer receives the ciphertexts from the previous mixer (or, in the case of the first mixer, from the bulletin board) and sends *shuffled* (permuted and rerandomized) ciphertext to the next mixer. Finally, only the output of the last mixer is decrypted. Assuming that at least one mixer is honest, it will be impossible to associate the decrypted votes to the voters that gave the original ciphertexts.

However, observe that a malicious mixer could easily switch out the ciphertexts and thus break the integrity of the election outcome. We can avoid such behaviour by requiring each mixer to provide a proof that the shuffling was done correctly. Additionally, to still maintain voters' privacy, this proof should not reveal any[1] information about the permutation or ciphertext randomizers used in the shuffle. This can be achieved with a *zero-knowledge (ZK) shuffle argument*.

Many efficient interactive arguments [FS01,TW10,Gro10,BG12] are known for shuffling, but interaction is not preferable in practice. For instance, we might want to audit elections months after it occurred, but mixers storing the private information might not be available anymore. Hence a better solution would be a non-interactive zero-knowledge (NIZK) argument, where the prover outputs a single message which can be later verified by anyone. Most interactive shuffle arguments can be made non-interactive using the Fiat-Shamir heuristic [FS87], but this only guarantees security in the random oracle model (ROM), where there are known cases in which the resulting argument is insecure [GK03,BDG$^+$13,BBH$^+$19].

As an alternative, the *Common Reference String (CRS)* model assumes a trusted party that samples a public string from some predefined distribution and provides it to both the prover and the verifier. In recent years several NIZK shuffle arguments have been proposed in this model [GL07,LZ13,GR16,FL16,FLZ16,FLSZ17a,FFHR19] that do not need ROM[2]. Arguably, the most practical proposal among these is the construction of Fauzi et al. [FLSZ17a][3], which we refer to as FLSZ throughout the text – it has comparable efficiency to interactive arguments and uses a standard ElGamal cryptosystem. However, a drawback of the CRS model is that it is unclear who should produce the CRS in practice. Sampling the CRS incorrectly, or even just leaking some side information (e.g., the simulation trapdoor), typically breaks the security of the argument. Several works have tried to alleviate this issue.

The *Bare Public Key (BPK)* model [CGGM00] requires significantly less trust than the CRS model. It removes the CRS and only requires the verifier to register a public key in a publicly accessible file before the protocol has started. A malicious verifier may choose the public key in any way she likes. However, BPK model NIZK with a standard auxiliary input ZK property can be cast as a

[1] Actually since the argument presented in this paper is statistically but not perfectly zero-knowledge, then it can leak information, but only with negligible probability.

[2] Even most of the interactive shuffle arguments require a CRS, but typically they have a less complicated structure and a uniformly random string usually suffices.

[3] The full version [FLSZ17a] mentions a security flaw in the conference version [FLSZ17b]. We follow the full version.

two-round ZK protocol, which is known to be impossible [GO94]. On the positive side, Wee [Wee07] has shown that BPK model NIZK is possible for a weaker non-uniform ZK. More recently, [ALSZ18] shows that NIZK with a related notion called no-auxiliary-string non-black-box ZK is also possible.

From a different perspective, Ben-Sasson et al. [BCG$^+$15] proposed a secure multi-party computation (MPC) protocol for CRS generation to distribute trust requirements. Essentially it is a distributed key generation (DKG) protocol that is secure if at least one party is honest. However, that protocol requires the ROM and only works for CRS-s with a very specific structure. Hence, it cannot be used as a black box, say, for the FLSZ argument. Subsequently, Abdolmaleki et al. [ABL$^+$19b], proposed a UC-secure variant of the Ben-Sasson et al.'s protocol which avoids the ROM by using a DL-extractable UC-commitment [ABL$^+$19a].

A series of results [BFS16,ABLZ17,Fuc18] have shown that CRS-based NIZK arguments can satisfy *subversion-ZK (Sub-ZK)*, i.e., the argument's ZK property holds even if the CRS is generated by an untrusted party. In particular, it has been shown [ABLZ17,Fuc18] that many existing *succinct non-interactive arguments of knowledge (SNARKs)* can be enhanced with a CRS verification algorithm CV, such that if $\mathsf{CV}(\mathsf{crs})$ accepts, then the proof will not leak any (non-negligible) information. So far, there is no general transformation which would give Sub-ZK property to any NIZK argument, and each new argument needs to be studied separately. Finally, recent work by Abdolmaleki et al. [ALSZ18] establishes a straightforward connection between Sub-ZK NIZK in the CRS model and BPK NIZK. Namely, a Sub-ZK NIZK can be transformed into a BPK NIZK (with non-auxiliary-input non-black-box ZK) where the verifier uses the CRS as her public key. This is also the direction we take in this paper as the BPK model is a more established and better-understood notion.

Our Contribution. We propose a new shuffle argument that we call a *transparent* FLSZ (denoted tFLSZ) which builds upon the result of [FLSZ17a] by significantly reducing the trust requirements, using weaker security assumptions, and also having a somewhat less complex structure.

FLSZ contains four subarguments: (i) a unit vector argument for showing that a committed message is a unit vector, i.e., a binary vector with exactly one 1, (ii) a permutation matrix argument for showing that n committed vectors form a permutation matrix, (iii) a same-message argument for showing that two committed vectors are equal, and (iv) a consistency argument for showing that the ciphertexts are shuffled according to the committed permutation matrix. However, in their case (i) the unit vector argument is not sound unless one also provides a related same-message argument and (ii) the consistency argument is only culpably sound, that is, soundness only holds against adversaries that can provide a witness of their cheating.

In tFLSZ, we combine the unit vector argument and the same-message argument into a new unit vector argument and prove its knowledge-soundness in the *al-*

gebraic group model (AGM) [FKL18] which is a weaker model compared to the generic bilinear group model (GBGM) used in [FLSZ17a]. Roughly speaking, in the GBGM an adversary is only allowed to perform group operations using an oracle which hides the actual structure of the group elements. On the other hand, the AGM allows the adversary to freely use the actual representation of elements in the group. Therefore security proofs in the AGM are usually reductions to some known assumption rather than unconditional proofs as in the GBGM. We show that knowledge-soundness of our new unit vector argument can be reduced to a quite standard q-type assumption in the algebraic group model.

The permutation argument is proven knowledge-sound assuming that the commitment scheme is binding and the unit vector argument is knowledge-sound. This again is a much weaker assumption compared to [FLSZ17a], where the authors prove a similar result but in the GBGM. Finally, we skip the consistency argument altogether, and directly prove that the shuffle argument is sound given that the permutation argument is knowledge-sound and that a variant of the *Kernel Matrix Diffie-Hellman (KerMDH)* assumption holds. We call this variant GapKerMDH and prove that in the AGM, it also reduces to the previously mentioned q-type assumption. The GapKerMDH assumption is weaker compared to the auxiliary-input KerMDH assumption used in [FLSZ17a] for their consistency argument. Interestingly, after simplifying the structure, the unit vector argument is the only subargument which depends on the AGM; the rest of the protocol is based on falsifiable assumptions [Nao03], i.e., assumptions where a challenger can efficiently verify that an adversary breaks the assumption (e.g., in the discrete logarithm assumption the challenger sends g^x, the adversary responds with x', and the challenger checks if $x = x'$). Falsifiable assumptions are much better understood and thus usually preferred over non-falsifiable assumptions such as knowledge assumptions [Dam92].

Secondly (and perhaps more importantly), we apply the efficient DKG protocol of Abdolmaleki et al. [ABL$^+$19b] which takes us from a setting of completely trusting the setup generator to a setting where we need to trust only one out of k parties in DKG. The modification, however, turns out to be non-trivial. We start by observing that the CRS of FLSZ is outside of the class of *verification-friendly* CRS-s that the DKG protocol can generate. Hence, in addition to simplifying the structure of FLSZ we also modify the CRS and make it verification-friendly, which mostly involves adding some well-chosen elements to the CRS. These additional elements are not needed for the honest prover or verifier but are available to dishonest parties. Therefore, after the DKG protocol finishes, these new CRS elements can be stored somewhere (in case someone wants to verify them in the future) and the effective CRS size (i.e., the size of the CRS used in the actual computation) does not change at all. If there is no need for transcript verification in the future, these additional elements can be safely disregarded after the computations are done. Hence, the CRS size in practice stays the same, but the security proofs must now consider a more powerful adversary.

As mentioned, the DKG protocol guarantees security (soundness and zero-knowledge) if at least one honest party participated. We take it one step further and prove that the protocol is also secure in the BPK model, following the ideas of [ALSZ18]. Namely, we construct a public key verification algorithm V_{pk} that the prover runs before outputting an argument. If V_{pk} is satisfied, then zero-knowledge holds even if the public key was generated by a single malicious party, or equivalently, if all of the parties in the DKG protocol colluded. However, if V_{pk} rejects the key, then the prover simply declines to output anything.

In Table 1 we compare efficiency and assumptions of the state-of-the-art non-interactive shuffle arguments. The argument by Groth [Gro10] has the best efficiency, but requires ROM and a trusted random string[4]. It is also worth to mention the argument by Bayer and Groth [BG12] which has sublinear communication but otherwise has the same drawbacks as [Gro10]. The argument of González and Ráfols [GR16] (and the slight improvement in [DGP$^+$19]) is based solely on falsifiable assumptions, but requires a quadratic size CRS which is not efficient enough for many applications. Similarly, Faonio et al. [FFHR19] use falsifiable assumptions but require pairings for all operations, making it inefficient. The Fauzi et al. [FLSZ17a] construction can be seen as a compromise between [Gro10] and [GR16]: efficiency is only slightly worse than [Gro10], does not require ROM, but some subarguments are proven in the GBGM. Our work retains almost the same efficiency as [FLSZ17a] by only adding n group elements to the CRS (we do not count elements solely needed by the DKG), but we make a significant reduction in the trust requirements for the setup phase and also prove security under weaker assumptions.

In summary, our new NIZK shuffle argument has the following properties:

1. Soundness holds assuming at least one honest party participated in the distributed key generation protocol and zero-knowledge holds even if all the parties were malicious.
2. Compared to the most-efficient shuffle argument without ROM [FLSZ17a]:
   (a) We simplify the structure of the argument.
   (b) We improve the security assumptions and isolate the unit vector argument as the only subargument which requires AGM.
   (c) The efficiency of the argument remains essentially the same.

Additionally, we implement our solution in Python 3.5+. See Section 7 for details.

2 Preliminaries

Let λ denote the security parameter. We write $f(\lambda) \approx_\lambda 0$, if a function f is negligible in λ. PPT stands for probabilistic polynomial time. We write $(a, b) \leftarrow$

[4] Namely, [Gro10] requires a commitment key for the extended Pedersen commitment which could be obtained from a uniformly random string

Table 1. Comparison of state-of-the-art shuffles. Exp. stands for exponentiations, pair. for pairings, n is the number of input ciphertexts and m is the number of mixers. Constant terms are neglected, shuffling is included to prover's efficiency, and shuffled ciphertexts are included to proof size.

	Prover efficiency	**Verifier efficiency**	**Decryption efficiency**	**Proof size**	**CRS size**	**Reference string**	**Assumptions**
[Gro10]	$8n$ exp.	$6n$ exp.	n exp.	$3n \times \mathbb{Z}_p$, $2n \times \mathbb{G}$	$n \times \mathbb{G}$	Uniform	ROM, DDH
[GR16]	$13n$ exp.	$13n$ pair.	n exp.	$4n \times \mathbb{G}_1$, $2n \times \mathbb{G}_2$	$(n^2+24n) \times \mathbb{G}_1, 23n \times \mathbb{G}_2$	Structured	Falsifiable
[FLSZ17a]	$11n$ exp.	$7n$ exp., $3n$ pair.	n exp.	$4n \times \mathbb{G}_1$, $3n \times \mathbb{G}_2$	$4n \times \mathbb{G}_1$, $n \times \mathbb{G}_2$	Structured	GBGM
[FFHR19]	$72n$ exp., $5n$ pair.	$22n$ pair.	$2n$ exp., $46n$ pair.	$12n \times \mathbb{G}_1$, $11n \times \mathbb{G}_2$, $4n \times \mathbb{G}_T$	$2m \times \mathbb{G}_1$, $2m \times \mathbb{G}_2$	Uniform	Falsifiable
This work	$11n$ exp.	$7n$ exp., $3n$ pair.	n exp.	$4n \times \mathbb{G}_1$, $3n \times \mathbb{G}_2$	$5n \times \mathbb{G}_1$, $n \times \mathbb{G}_2$	Verifiable	AGM

$(\mathcal{A} \| \mathsf{Ext})(x)$ if algorithms $\mathcal{A}$ and Ext on the same input x and random tape r output $a \leftarrow \mathcal{A}(x;r)$ and $b \leftarrow \mathsf{Ext}(x;r)$. By $\mathsf{RND}(\mathcal{A})$ we denote the random tape of $\mathcal{A}$ and by $\mathsf{Range}(\mathcal{A}(x))$ the set of all possible outputs of $\mathcal{A}$ given input x.

We write $x \leftarrow_{\$} A$ if x is sampled uniformly randomly from the set A. By default $\boldsymbol{x} = (x_i)_{i=1}^n \in A^n$ is a column vector and $\mathbf{1}_n := (1)_{i=1}^n$, $\mathbf{0}_n := (0)_{i=1}^n$. A set of permutations on n elements is denoted by $\mathbb{S}_n$. A matrix $\boldsymbol{A} \in \{0,1\}^{n \times n}$ is a permutation matrix of the permutation $\sigma \in \mathbb{S}_n$ when $A_{i,j} = 1$ iff $\sigma(i) = j$. We call $\boldsymbol{a}$ a unit vector if it contains exactly one 1 and all other positions are 0. Let $\mathbb{F}_p$ be a finite field of prime order p and $\mathbb{F}_p^* := \mathbb{F}_p \backslash \{1\}$. For vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_p^n$, $\boldsymbol{x} \circ \boldsymbol{y}$ denotes the entry-wise product. We use the bracket notation where $[x]$ denotes the group element with discrete logarithm x. We consider additive groups, thus $[a] + [b] = [a+b]$. For integers $a < b$ we denote $[a \mathinner{..} b] := \{a, a+1, \ldots, b\}$.

Bilinear Pairing. A bilinear group generator $\mathsf{BGen}(1^\lambda)$ outputs a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathcal{P}_1, \mathcal{P}_2, \bullet)$ such that (i) p is a prime of length $\Theta(\lambda)$, (ii) for $k \in \{1,2\}$, $\mathbb{G}_k$ is an additive group of order p with a generator $\mathcal{P}_k$, and (iii) $\bullet$ is a map $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We set $\mathcal{P}_T := \mathcal{P}_1 \bullet \mathcal{P}_2$ and use the bracket notation by defining $[a]_k := a \cdot \mathcal{P}_k$, for $k \in \{1, 2, T\}$. We require that

- $[a]_1 \bullet [b]_2 = [ab]_T$ for all $a, b \in \mathbb{F}_p$ (*bilinearity*),
- $\mathcal{P}_T \neq [0]_T$ (*non-degeneracy*), and
- $\bullet$ is efficiently computable.

In the following we use *asymmetric* bilinear groups where there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. For the state of the art in pairing constructions see [BD17].

Bracket notation extends naturally to matrices and vectors, e.g., we may write $[\boldsymbol{A}]_1 \bullet [\boldsymbol{B}]_2 = [\boldsymbol{I}]_1 \bullet (\boldsymbol{A}[\boldsymbol{B}]_2) = [\boldsymbol{I}]_1 \bullet [\boldsymbol{AB}]_2$ for $\boldsymbol{A} \in \mathbb{F}_p^{n \times m}$, $\boldsymbol{B} \in \mathbb{F}_p^{m \times k}$, and identity matrix $\boldsymbol{I} \in \mathbb{F}_p^{n \times n}$. Occasionally we write $[a]_z$ for $z \in \{1, 2\}$ and use $\bar{z} := 3 - z$ to denote the number of the other non-target group. Then $[a]_z \bullet [b]_{\bar{z}}$ would mean $[a]_1 \bullet [b]_2$ for $z = 1$ and $[b]_1 \bullet [a]_2$ for $z = 2$.

Lagrange basis. Let $\omega_1, \ldots, \omega_{n+1}$ be distinct points in $\mathbb{F}_p$. For $i \in [1 \mathinner{..} n+1]$, the i-th Lagrange basis polynomial is defined as $\ell_i(X) := \prod_{j \neq i} \frac{X - \omega_j}{\omega_i - \omega_j}$. Hence, it is the unique degree n polynomial such that $\ell_i(\omega_i) = 1$ and $\ell_i(\omega_j) = 0$ for all $j \neq i$. As the name suggests, $\{\ell_i(X)\}_{i=1}^{n+1}$ is a basis for $\{f \in \mathbb{F}_p[X] : \deg(f) \leq n\}$.

Encryption scheme. A public key encryption scheme is a triple of PPT algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that

- $\mathsf{KGen}(1^\lambda)$ outputs a public key and a secret key pair $(\mathsf{pk_e}, \mathsf{sk_e})$.
- $\mathsf{Enc}_{\mathsf{pk_e}}(m; r)$ outputs a ciphertext c encrypting the message m with randomness r under the public key $\mathsf{pk_e}$.
- $\mathsf{Dec}_{\mathsf{sk_e}}(c)$ outputs the decryption of the ciphertext c using the secret key $\mathsf{sk_e}$.

We require that $\mathsf{Dec}_{\mathsf{sk_e}}(\mathsf{Enc}_{\mathsf{pk_e}}(m; r)) = m$ for every message m and randomizer r. Intuitively, an encryption scheme is *IND-CPA*-secure if no PPT adversary $\mathcal{A}$ can distinguish between the ciphertext distributions of any two messages.

We use the ElGamal encryption scheme over a group $\mathbb{G}_2$ defined as follows. The algorithm $\mathsf{KGen}(1^\lambda)$ samples $\mathsf{sk_e} \leftarrow_{\$} \mathbb{F}_p$ and outputs $(\mathsf{pk_e} := [1, \mathsf{sk_e}]_2, \mathsf{sk_e})$. An encryption of a message $[m]_2$ is $\mathsf{Enc}_{\mathsf{pk_e}}([m]_2; r) := [0, m]_2 + r \cdot \mathsf{pk_e}$ where $r \leftarrow_{\$} \mathbb{F}_p$. A ciphertext $[\boldsymbol{c}]_2 = [c_1, c_2]_2$ is decrypted by computing $\mathsf{Dec}_{\mathsf{sk_e}}([\boldsymbol{c}]_2) := [c_2]_2 - \mathsf{sk_e} \cdot [c_1]_2$. ElGamal is IND-CPA-secure if the DDH assumption holds in group $\mathbb{G}_2$. ElGamal is also *blindable*, meaning that $\mathsf{Enc}_{\mathsf{pk_e}}([m]; r) + \mathsf{Enc}_{\mathsf{pk_e}}([0], r') = \mathsf{Enc}_{\mathsf{pk_e}}([m], r + r')$ and, assuming that $r' \leftarrow_{\$} \mathbb{F}_p$, no PPT adversary can distinguish if $\mathsf{Enc}_{\mathsf{pk_e}}([m]; r)$ and $\mathsf{Enc}_{\mathsf{pk_e}}([m]; r + r')$ encrypt the same message or not.

Non-Interactive Zero-Knowledge. Let $\mathcal{R} = \{(\mathsf{x}, \mathsf{w})\}$ be a relation such that $\mathcal{L}_{\mathcal{R}} = \{\mathsf{x} : \exists \mathsf{w}\ (\mathsf{x}, \mathsf{w}) \in \mathcal{R}\}$ is an NP language where w is a witness for x. Following [ALSZ18], we define a NIZK argument in the BPK model as follows.

A NIZK argument Ψ in the BPK model for relation $\mathcal{R}$ is a tuple efficient algorithms $(\mathsf{Pgen}, \mathsf{K_{td}}, \mathsf{K_{pk}}, \mathsf{V_{pk}}, \mathsf{P}, \mathsf{V}, \mathsf{Sim})$, where

- $\mathsf{Pgen}(1^\lambda)$ is a deterministic algorithm that outputs a setup parameter gk.
- $\mathsf{K_{td}}(\mathsf{gk})$ is a PPT algorithm that on input gk outputs a trapdoor td.
- $\mathsf{K_{pk}}(\mathsf{gk}, \mathsf{td})$ is a deterministic algorithm that on input gk and $\mathsf{td} \in \mathsf{Range}(\mathsf{K_{td}}(\mathsf{gk}))$ outputs a public key pk.
- $\mathsf{V_{pk}}(\mathsf{gk}, \mathsf{pk})$ is a PPT algorithm that on input gk and a public key pk outputs 0 (if the key is malformed) or 1 (if the key is well-formed).
- $\mathsf{P}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \mathsf{w})$ is a PPT algorithm that given a setup parameter gk, public key pk, and $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, outputs an argument π.

- $\mathsf{V}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \pi)$ is a PPT algorithm that on input a setup parameter gk, public key pk, statement x, and argument π, outputs 0 (reject) or 1 (accept).
- $\mathsf{Sim}(\mathsf{gk}, \mathsf{pk}, \mathsf{td}, \mathsf{x})$ is a PPT algorithm that on input a setup parameter gk, public key pk, trapdoor td, and $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$ outputs a simulated argument π.

For the sake of brevity, we sometimes use the algorithm $\mathsf{K}(\mathsf{gk}) := \mathsf{K}_{\mathsf{pk}}(\mathsf{gk}, \mathsf{K}_{\mathsf{td}}(\mathsf{gk}))$. By a *NIZK argument in the CRS model* we mean a tuple $(\mathsf{Pgen}, \mathsf{K}_{\mathsf{td}}, \mathsf{K}_{\mathsf{pk}}, \mathsf{P}, \mathsf{V}, \mathsf{Sim})$ of the above algorithms (i.e., all except V_{pk}).

Completeness simply requires that an honestly generated key and argument are respectively accepted by V_{pk} and V. We give the definition for the BPK model. The definition for the CRS model neglects the condition $\mathsf{V}_{\mathsf{pk}}(\mathsf{gk}, \mathsf{pk}) = 1$.

Definition 1. *The argument Ψ in BPK model is* perfectly complete *if for all λ, and $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, the following probability is 1,*

$$\Pr\left[\mathsf{gk} \leftarrow \mathsf{Pgen}(1^\lambda), \mathsf{pk} \leftarrow \mathsf{K}(\mathsf{gk}) : \mathsf{V}_{\mathsf{pk}}(\mathsf{gk}, \mathsf{pk}) = 1 \wedge \mathsf{V}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \mathsf{P}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \mathsf{w})) = 1\right] .$$

Soundness guarantees that a malicious prover cannot create a valid argument for a false statement. The definitions match in the BPK model and the CRS model.

Definition 2. *The argument Ψ is* sound *if for any PPT adversary $\mathcal{A}$,*

$$\Pr\begin{bmatrix}\mathsf{gk} \leftarrow \mathsf{Pgen}(1^\lambda), (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{K}(\mathsf{gk}), (\mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathsf{gk}, \mathsf{pk}) : \\ \mathsf{x} \notin \mathcal{L}_{\mathcal{R}} \wedge \mathsf{V}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \pi) = 1\end{bmatrix} \approx_\lambda 0.$$

Knowledge-soundness strengthens the previous definition by requiring that the prover "knows" the witness, i.e., there exists an extractor that outputs the witness given the code and random coins of the adversary.

Definition 3. *The argument Ψ is* knowledge-sound *if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor* Ext*, such that*

$$\Pr\begin{bmatrix}\mathsf{gk} \leftarrow \mathsf{Pgen}(1^\lambda), (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{K}(\mathsf{gk}), ((\mathsf{x}, \pi), \mathsf{w}) \leftarrow (\mathcal{A} \| \mathsf{Ext})(\mathsf{gk}, \mathsf{pk}) : \\ (\mathsf{x}, \mathsf{w}) \notin \mathcal{R} \wedge \mathsf{V}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \pi) = 1\end{bmatrix} \approx_\lambda 0.$$

Lastly, zero-knowledge guarantees that the argument leaks no information besides that $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$ by giving an algorithm Sim which, given a trapdoor, can create a valid argument for any $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$ without knowing the corresponding witness.

Definition 4. *An argument Ψ in the CRS model is* statistically *zero-knowledge, if for any adversary $\mathcal{A}$, and any $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, $\varepsilon_0 \approx_\lambda \varepsilon_1$, where*

$$\varepsilon_b := \Pr\begin{bmatrix}\mathsf{gk} \leftarrow \mathsf{Pgen}(1^\lambda), (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{K}(\mathsf{gk}), \textbf{if } b = 0 \textbf{ then } \pi \leftarrow \mathsf{P}(\mathsf{gk}, \mathsf{crs}, \mathsf{x}, \mathsf{w}) \\ \textbf{else } \pi \leftarrow \mathsf{Sim}(\mathsf{gk}, \mathsf{crs}, \mathsf{td}, \mathsf{x}) \textbf{ fi} : \mathcal{A}(\mathsf{gk}, \mathsf{crs}, \pi) = 1\end{bmatrix} .$$

We say that Ψ is perfectly zero-knowledge *if $\varepsilon_0 = \varepsilon_1$.*

In the BPK model, we use the *no-auxiliary-string non-black-box zero-knowledge* definition of [ALSZ18] (as mentioned, NIZK is impossible with the standard BPK ZK definition). Essentially the prover first runs a public key verification algorithm V_{pk} to check well-formedness of the key pk and only then outputs a proof. Compared to the previous definition, we require that there exists an extractor that extracts a trapdoor for any well-formed pk given access to adversary's random coins. Intuitively this guarantees that the key generator knows the trapdoor and thus could generate the proof himself using the simulator.

Definition 5 ([ALSZ18]). *The argument Ψ in the BPK model is* statistically no-auxiliary-string non-black-box zero-knowledge *(nn-ZK), if for any PPT subverter* X *there exists a PPT extractor* $\mathsf{Ext}_{\mathsf{X}}$*, s.t., for any (stateful) adversary $\mathcal{A}$, $\varepsilon_0 \approx_\lambda \varepsilon_1$, where*

$$\varepsilon_b := \Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{Pgen}(1^\lambda), (\mathsf{pk}, \mathsf{aux}_{\mathsf{X}} \| \mathsf{td}) \leftarrow (\mathsf{X} \| \mathsf{Ext}_{\mathsf{X}})(\mathsf{gk}), (\mathsf{x}, \mathsf{w}) \leftarrow \mathcal{A}(\mathsf{aux}_{\mathsf{X}}), \\ \textbf{if } b = 0 \textbf{ then } \pi \leftarrow \mathsf{P}(\mathsf{gk}, \mathsf{pk}, \mathsf{x}, \mathsf{w}) \textbf{ else } \pi \leftarrow \mathsf{Sim}(\mathsf{gk}, \mathsf{pk}, \mathsf{td}, \mathsf{x}) \textbf{ fi} : \\ (\mathsf{x}, \mathsf{w}) \in \mathcal{R} \wedge \mathsf{V}_{\mathsf{pk}}(\mathsf{gk}, \mathsf{pk}) = 1 \wedge \mathcal{A}(\pi) = 1\end{array}\right].$$

Here $\mathsf{aux}_{\mathsf{X}}$ *is whatever information* X *wishes to send to $\mathcal{A}$.*

Assumptions. In AGM reductions we use q-PDL, a q-type version of discrete logarithm assumption. We also require the KerMDH computational assumption, and the BDH-KE knowledge assumption. The definitions are as follows.

Definition 6 (q-PDL [Lip12]). *The q-Power Discrete Logarithm assumption holds for* BGen *if for any PPT $\mathcal{A}$,*

$$\Pr[\mathsf{gk} \leftarrow \mathsf{BGen}(1^\lambda), z \leftarrow_\$ \mathbb{Z}_p, z' \leftarrow \mathcal{A}(\mathsf{gk}, [(z^i)_{i=1}^q]_1, [(z^i)_{i=1}^q]_2) : z = z'] \approx_\lambda 0.$$

Definition 7 (KerMDH [MRV16]). *Let $\mathcal{D}_{\ell,k}$ be a distribution over $\mathbb{F}_p^{\ell \times k}$. The $\mathcal{D}_{\ell,k}$-KerMDH assumption holds for* BGen *and $z \in \{1, 2\}$ if for any PPT $\mathcal{A}$,*

$$\Pr[\mathsf{gk} \leftarrow \mathsf{BGen}(1^\lambda), \boldsymbol{M} \leftarrow_\$ \mathcal{D}_{\ell,k}, [\boldsymbol{c}]_{\bar{z}} \leftarrow \mathcal{A}(\mathsf{gk}, [\boldsymbol{M}]_z) : \boldsymbol{c} \neq \boldsymbol{0} \wedge \boldsymbol{c}^\top \boldsymbol{M} = \boldsymbol{0}] \approx_\lambda 0.$$

Definition 8 (BDH-KE [ABLZ17]). *We say that* BGen *is BDH-KE secure if for any PPT adversary $\mathcal{A}$ there exists a PPT extractor* $\mathsf{Ext}_{\mathcal{A}}$*, such that*

$$\Pr\left[\mathsf{gk} \leftarrow \mathsf{BGen}(1^\lambda), ([\alpha]_1, [\alpha']_2 \| \beta) \leftarrow (\mathcal{A} \| \mathsf{Ext}_{\mathcal{A}})(\mathsf{gk}) : \alpha = \alpha' \wedge \beta \neq \alpha \quad \right] \approx_\lambda 0.$$

Commitment Scheme. A commitment scheme is a tuple of efficient algorithms $(\mathsf{KGen}, \mathsf{Com})$ such that

– $\mathsf{KGen}(1^\lambda)$ outputs a commitment key ck.
– $\mathsf{Com}_{\mathsf{ck}}(m; r)$ outputs a commitment c given a message m and randomness r.

Typically a commitment scheme should satisfy at least the following properties.

- *(perfectly) hiding*: the distribution $\mathsf{Com}_{\mathsf{ck}}(m; r)$ (over $r \leftarrow_{\$} \mathbb{F}_p$) is the same for any message m;
- *(computationally) binding*: it is infeasible for an adversary to find (m_1, r_1) and (m_2, r_2) s.t. $\mathsf{Com}_{\mathsf{ck}}(m_1; r_1) = \mathsf{Com}_{\mathsf{ck}}(m_2; r_2)$ and $m_1 \neq m_2$.

Polynomial Commitment Scheme. For polynomials $\{T_i(X_1, \ldots, X_k)\}_{i=1}^{n+1} \in \mathbb{F}_p[X_1, \ldots, X_k]$ we define a $(T_i)_{i=1}^{n+1}$-*commitment scheme* as follows:

- $\mathsf{KGen}(1^\lambda)$ picks $\boldsymbol{\chi} \leftarrow_{\$} \mathbb{F}_p^k$ and returns a commitment key $\mathsf{ck} \leftarrow \left[(T_i(\boldsymbol{\chi}))_{i=1}^{n+1}\right]_z$.
- $\mathsf{Com}_{\mathsf{ck}}((a_1, \ldots, a_n); r)$ returns a commitment $\sum_{i=1}^{n} a_i [T_i(\boldsymbol{\chi})]_1 + r[T_{n+1}(\boldsymbol{\chi})]_1$.

Clearly, this commitment is perfectly hiding when $r \leftarrow_{\$} \mathbb{F}_p$ and $T_{n+1}(\boldsymbol{\chi}) \neq 0$. If $\{T_i\}_{i=1}^{n+1}$ is a linearly independent set, it is also computationally binding under a suitably chosen KerMDH assumption, cf. [FLSZ17a, Theorem 1].

DL-Extractable Commitment Scheme. The DKG protocol of [ABL$^+$19b] requires a UC-secure *Discrete Logarithm Extractable* (DL-extractable) commitment scheme as defined in [ABL$^+$19a]. In DL-extractable commitments the messages are field elements x, but commitments can be opened to $[x]_z$ thus still leaving x itself private. However, since in the UC-model committing to x is equivalent to giving it to an ideal functionality, then the committer knows x, i.e., the discrete logarithm x can be extracted from the commitment given a secret key. For a formal definition and a construction, see [ABL$^+$19a].

Algebraic Group Model. Recently Fuchsbauer et al. [FKL18] introduced the algebraic group model (AGM) that lies between the standard and the generic group model. In the AGM, an adversary $\mathcal{A}$ that returns a group element $[x]_z$ is required to provide a linear representation of $[x]_z$ relative to all previously received group elements. That is, if $\mathcal{A}$ received as input group elements $[\boldsymbol{y}]_z$ then she must submit along with $[x]_k$ a representation $\boldsymbol{z}$ such that $[x]_z = \boldsymbol{z}^\top [\boldsymbol{y}]_z$. Using techniques similar to [FKL18, Theorem 7.2] we prove knowledge-soundness of the unit vector argument under the PDL assumption in the AGM.

2.1 FLSZ Shuffle Argument

We give a brief overview of the FLSZ shuffle argument for the shuffle relation

$$\mathcal{R}_n^{sh} := \left\{ \begin{array}{l} \left((\mathsf{gk}, \mathsf{pk}_{\mathsf{e}}, [(\boldsymbol{c}_i')_{i=1}^n]_2, [(\boldsymbol{c}_i)_{i=1}^n]_2), (\sigma, \boldsymbol{t})\right) \mid \sigma \in \mathbb{S}_n \wedge \boldsymbol{t} \in \mathbb{F}_p^n \wedge \\ \left(\forall i \in [1 \mathinner{..} n] : [\boldsymbol{c}_i']_2 = [\boldsymbol{c}_{\sigma(i)}]_2 + \mathsf{Enc}_{\mathsf{pk}_{\mathsf{e}}}([0]_2; t_i)\right) \end{array} \right\} .$$

They use a $((P_i(X))_{i=1}^n, X_\varrho)$-commitment scheme to commit to columns of a permutation matrix, where $P_i(X) := 2\ell_i(X) + \ell_{n+1}(X)$ for $i \in [1 \mathinner{..} n]$.

Lemma 1. *Let* $P_0(X) := \ell_{n+1}(X) - 1$ *and* $Q_i(X) := (P_i(X) + P_0(X))^2 - 1$ *for* $i \in [1\,..\,n]$. *If* $(\sum_{i=1}^{n} a_i P_i(X) + P_0(X))^2 - 1 \in \mathsf{Span}\{Q_i(X)\}_{i=1}^{n}$ *and* $n < p - 1$, *then* $(a_1, \dots, a_n)$ *is a unit vector.*

Proof. Denote $T(X) := (\sum_{i=1}^{n} a_i P_i(X) + P_0(X))^2 - 1$. Firstly, observe that for $j \in [0\,..\,n]$, $T(w_j) = (\sum_{i=1}^{n} a_i P_i(\omega_j) + P_0(\omega_j))^2 - 1 = (\sum_{i=1}^{n} a_i(2\ell_i(\omega_j) + \ell_{n+1}(\omega_j)) + \ell_{n+1}(\omega_j) - 1)^2 - 1 = (2a_j - 1)^2 - 1 = 4a_j(a_j - 1)$. On the other hand, $Q_i(\omega_j) = (P_i(\omega_j) + P_0(\omega_j))^2 - 1 = 0$ for $j \in [1\,..\,n]$. Therefore, $T(X) \in \mathsf{Span}\{Q_i(X)\}_{i=1}^{n}$ implies that $T(\omega_j) = 0$. Hence $a_j \in \{0, 1\}$ for $j \in [1\,..\,n]$.

Finally, $T(\omega_{n+1}) = (\sum_{i=1}^{n} a_i(2 \cdot 0 + 1) + 1 - 1))^2 - 1 = (\sum_{i=1}^{n} a_i)^2 - 1$. Similarly as before, $Q_i(\omega_{n+1}) = 0$ so $T(w_{n+1}) = 0$. Therefore, $(\sum_{i=1}^{n} a_i)^2 - 1 = (\sum_{i=1}^{n} a_i - 1)(\sum_{i=1}^{n} a_i + 1) = 0$. Since $\sum_{i=1}^{n} a_i = n < p - 1$ we must have $\sum_{i=1}^{n} a_i = 1$, so exactly one a_j is 1 and all others are 0. Hence $(a_1, \dots, a_n)$ is a unit vector. $\square$

Given the above property, they propose a *unit vector argument* to show that the prover could open each commitment to a unit vector. They then enhance it to a *permutation matrix argument* by observing that n unit vectors form a permutation matrix exactly when their sum is $\mathbf{1}_n$. Next, they would like to show that the committed permutation matrix was used to shuffle the ciphertexts. However, due to some technical challenges, they are unable to use the same commitment key. Instead, they commit once more to the columns of the permutation matrix, but this time with a $((\hat{P}_i(X))_{i=1}^{n}, X_{\hat{\varrho}})$-commitment where $\hat{P}_i(X) := X^{(i+1)(n+1)}$ for $i \in [1\,..\,n]$. They propose a *same-message argument* to show that both types of commitments can be opened to the same matrix. Finally, a *consistency argument* proves that the committed permutation was used to shuffle the ciphertexts.

The unit vector argument, the permutation matrix argument, and the same-message argument are proven to be knowledge-sound in the GBGM. However, the soundness of the unit vector argument depends on the soundness of the same-message argument. The consistency argument is culpably sound[5] under an application specific variation of the KerMDH assumption. The shuffle argument itself is sound assuming that other arguments are secure and assuming that commitments are binding. The shuffle argument has perfect zero-knowledge.

3 Distributed Key Generation Protocol

We apply the UC-secure DKG protocol of Abdolmaleki et al. [ABL+19b] in the public key generation of our shuffle argument. This protocol avoids the random oracle model (unlike, e.g., [BCG+15]) and due to UC-security it will not affect the soundness or zero-knowledge properties of the argument. Of course, any general MPC protocol can be used as a DKG, but since we potentially require

[5] Culpable soundness is a weaker form of soundness where an adversary additionally provides a witness of his cheating.

a large number of parties (e.g., mixers in the mix-network) and since evaluated circuits can have a large multiplicative depth, specialized protocols will perform much better. See [BCG+15] for further discussion on efficiency difference.

3.1 Verification-Friendly Public Key

Although the DKG protocols of [BCG+15] and [ABL+19b] are efficient, they are not general MPC protocols and can only generate certain kinds of keys. Namely, they require key computation to be represented as a circuit that comes from a special class ($\mathcal{C}^{\mathsf{S}}$, described below) and is evaluated on uniformly random field inputs. Fortunately, the protocols are still sufficient for generating public keys for many pairing-based arguments or, as we will later show, slightly modified versions. Compared to [ABL+19b] we give a more direct, but equivalent, description of such keys which we call *verification-friendly.* Intuitively, a verification-friendly public key means that even if one doesn't trust the parties generating the public key, one can at least ensure that it is of the correct structure.

We say that an argument Ψ has a *verification-friendly public key* if (i) output $\mathsf{td} = (\chi_i)_{i=1}^n$ of $\mathsf{K}_{\mathsf{td}}(\mathsf{gk})$ is distributed uniformly randomly over $(\mathbb{F}_p^*)^n$, and (ii) $\mathsf{K}_{\mathsf{pk}}(\mathsf{gk}, \mathsf{td}) = \mathsf{C}(\mathsf{td})$ where C is a circuit from a class $\mathcal{C}^{\mathsf{S}}_{\mathsf{gk},n}$. Any circuit $\mathsf{C} \in \mathcal{C}^{\mathsf{S}}_{\mathsf{gk},n}$ takes as input $\mathsf{td} = (\chi_i)_{i=1}^n \in (\mathbb{F}_p^*)^n$ and contains two types of gates:

- *multiplication-division (multdiv)* gate $\mathsf{MD}_{\chi_i,\chi_j}([x]_z)$ outputs $[(\chi_i/\chi_j)x]_z$, where $z \in \{1, 2\}$ and $[x]_z$ is a gate input.
- *linear combination (lincomb)* gate $\mathsf{LC}_{\boldsymbol{c}}([\boldsymbol{y}]_z)$ outputs $\left[\sum_{i=1}^t c_i y_i\right]_z$, where $z \in \{1, 2\}$, $\boldsymbol{c} \in \mathbb{F}_p^t$ is a constant, and $[\boldsymbol{y}]_z \in \mathbb{G}_z^t$ is a gate input.

Gates in the circuit C are partitioned into interleaved layers $C_1, L_1, \ldots, C_d, L_d$ where each C_i contains only multdiv gates and L_i contains only lincomb gates. Furthermore, C satisfies the following conditions:

1. Inputs of gates in C_i or L_i can be either constants or outputs of the gates on the current or lower layers of the circuit.
2. The output of each gate is part of the output of the circuit C.
3. Layer C_1 always contains gates $\mathsf{MD}_{\chi_i,1}([1]_z)$ for all $i \in [1 \mathinner{..} n]$, $z \in \{1, 2\}$. Therefore, $[(\chi_i)_{i=1}^n]_1$ and $[(\chi_i)_{i=1}^n]_2$ are always outputs of the circuit.

3.2 DKG Protocol for Verification-Friendly Keys

We describe the DKG protocol of [ABL+19b] where the parties collectively evaluate a $\mathcal{C}^{\mathsf{S}}_{\mathsf{gk},n}$-circuit to generate a verification-friendly public key. The protocol retains soundness and zero-knowledge of the argument given that at least one party in the protocol is honest and malicious parties are non-halting. We note that with a suitable key verification algorithm it is possible to achieve zero-knowledge even if all the parties are malicious.

```
$\mathsf{mpcMD}_{\chi_i,\chi_j}([x]_z)$:
  1. Set $\mathsf{cert}_0 \leftarrow [x]_z$.
  2. For $r = 1, \ldots, k$: Party $\mathcal{P}_r$ broadcasts $\mathsf{cert}_r \leftarrow (\chi_{i,r}/\chi_{j,r}) \cdot \mathsf{cert}_{r-1}$.
  3. Output $\mathsf{cert}_k$.
```

```
$\mathsf{VmpcMD}_{\chi_i,\chi_j}([x]_z, (\mathsf{cert}_r)_{r=1}^k, [(\chi_{j,r})_{r=1}^k, (\chi_{i,r})_{r=1}^k]_{\bar{z}})$:
  1. Set $\mathsf{cert}_0 \leftarrow [x]_z$.
  2. For $r = 1, \ldots, k$: check that $\mathsf{cert}_r \bullet [\chi_{j,r}]_{\bar{z}} = \mathsf{cert}_{r-1} \bullet [\chi_{i,r}]_{\bar{z}}$.
  3. If all checks pass output 1 and otherwise output 0.
```

Fig. 1. Multi-party protocol $\mathsf{mpcMD}_{\chi_i,\chi_j}$ and its transcript verifier $\mathsf{VmpcMD}_{\chi_i,\chi_j}$

Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be the parties running the DKG protocol. Each party $\mathcal{P}_r$ samples shares $(\chi_{j,r})_{j=1}^n \leftarrow_\$ (\mathbb{F}_p^*)^n$ which allows us to define trapdoor elements as $\chi_j := \prod_{r=1}^k \chi_{j,r}$ for $j \in [1 \mathinner{..} n]$. Note that if at least one value $\chi_{j,r} \in \mathbb{F}_p^*$ is picked independently and uniformly at random, then χ_j is uniformly random in $\mathbb{F}_p^*$. For ease of description, we set $\chi_0 := 1$ and similarly $\chi_{0,r} := 1$ for $r \in [1 \mathinner{..} k]$.

The protocol starts with a commitment round where all the parties commit to their shares $\chi_{i,r}$ with a UC-secure DL-extractable commitment scheme. This is followed by an opening round where each $\mathcal{P}_i$ reveals $[\chi_{i,r}]_1, [\chi_{i,r}]_2$. Since the commitment scheme is UC-secure, then it is also non-malleable and thus guarantees that the adversary chooses her shares independently of the shares of the honest parties. Next, the parties start to evaluate the circuit layer-by-layer. For evaluating a single multdiv gate $\mathsf{MD}_{\chi_i,\chi_j}([x]_z) = [(\chi_i/\chi_j)x]_z$ where $i, j \in [0 \mathinner{..} n]$, parties run the $\mathsf{mpcMD}_{\chi_i,\chi_j}([x]_z)$ protocol given in Fig. 1. Assuming that $[x]_z$ is public, $\mathcal{P}_1$ broadcasts $(\chi_{i,1}/\chi_{j,1})\,[a]_z$ and each subsequent party $\mathcal{P}_r$ multiplies $\chi_{i,r}/\chi_{j,r}$ to the output of her predecessor $\mathcal{P}_{r-1}$. If all the parties follow the protocol, then the output of $\mathcal{P}_k$ is $\mathsf{cert}_k = (\chi_{i,1} \cdot \ldots \cdot \chi_{i,k})/(\chi_{j,1} \cdot \ldots \cdot \chi_{j,k})\,[a]_z = (\chi_i/\chi_j)\,[a]_z$. Computation of each party can be verified with pairings by using the algorithm $\mathsf{VmpcMD}_{\chi_i,\chi_j}$ in Fig. 1. Any linear combination gate $\mathsf{LC}_{\boldsymbol{c}}([\boldsymbol{x}]_z)$ can be computed locally by each party by simply evaluating the expression $\sum_{i=1}^t c_i\,[a_i]_z$.

Let us make a slight restriction for now that multdiv gates on the same layer do not depend on each other. Then each multi-division layer C_i can be evaluated by running multiple instances of the mpcMD protocol in parallel. More precisely, the computation begins with the party $\mathcal{P}_1$ doing its part of computation in mpcMD for each multdiv gate in C_i. Then, given the output produced by $\mathcal{P}_1$, the party $\mathcal{P}_2$ does her part of the computation for each gate in the layer C_i and so on. Hence, a single multdiv layer can be evaluated in k rounds since every party needs to contribute to the output of the previous party just once. After each multi-division layer, the parties verify the computation by running the algorithm $\mathsf{VmpcMD}_{\chi_i,\chi_j}$ for each gate. If the checks pass, the parties locally evaluate gates on layer L_i and proceed to compute the next layer C_{i+1}. Full details are given in Fig. 2.

Commitment: Each party $\mathcal{P}_r$ picks $\chi_{1,r}, \ldots, \chi_{n,r} \leftarrow\$ \mathbb{F}_p^*$ and broadcasts DL-extractable commitments of the values.
Opening: Once all the commitments are received, $\mathcal{P}_r$ broadcasts openings together with $[(\chi_{i,r})_{i=1}^n]_1$ and $[(\chi_{i,r})_{i=1}^n]_2$. Each party verifies the openings and aborts if the verification failed.
Layer computation: For a multi-division layer C_i containing a gate $\mathsf{MD}_{\chi_i,\chi_j}([a]_z)$, parties run the protocol $\mathsf{mpcMD}_{\chi_i,\chi_j}([a]_z)$ and verify the computation with the algorithm $\mathsf{VmpcMD}_{\chi_i,\chi_j}$. All the gates in C_i can be evaluated in parallel. Linear combination layers L_i are locally evaluated by each party.
Output: Output of the protocol is the output of all the evaluated gates.

Fig. 2. Distributed key generation protocol for a circuit $\mathsf{C} = (C_1, L_1, \ldots, C_d, L_d)$

We refer the reader to [ABL+19b] for the more general protocol where k rounds can be achieved even if the gates on the same layer depend on each other. That version of the DKG is also used for our shuffle argument, but for this we provide an explicit description in Appendix B. It is important to note that Abdolmaleki et al. showed that if at least one party in the DKG is honest, then it UC-realises the CRS ideal functionality (which essentially samples a public key in the beginning and returns it to anyone that queries).

4 Transparent Shuffle Argument

The DKG protocol requires the public key to be verification-friendly. In particular, we need to guarantee the following properties:

- Each trapdoor $\iota \in \mathsf{td}$ has to be sampled uniformly at random from $\mathbb{F}_p^*$ and the public key has to contain both $[\iota]_1$ and $[\iota]_2$.
- The public key has to be computable by interleaved multi-division and linear combination circuit layers and the output of each gate has to be part of the public key. For example, given that $[a]_1, [b]_1, [c]_1, [d]_1$ are part of the public key, it is not possible to have $[ab+cd]_1$ in the public key without also revealing some intermediate gate outputs like $[ab]_1$ and $[cd]_1$.

In this section, we modify the FLSZ argument and construct a new transparent shuffle argument tFLSZ which has a verification-friendly public key. Besides making the argument verification-friendly, we also simplify the construction: (i) we combine the unit vector argument and the same-message argument of tFLSZ into a single argument, (ii) we skip the consistency argument and directly construct a shuffle argument from the permutation argument, and (iii) we observe that one of the trapdoors, $\hat{\varrho}$, can be set to 1 without affecting security. The new argument is given in Fig. 3; we introduce the construction step-by-step in the following.

Let us take the public key of FLSZ in Fig. 4 as a starting point and observe which modifications need to be introduced to make it verification-friendly.

```
$\mathsf{K}_{\mathsf{td}}(\mathsf{gk})$: Return $\mathsf{td} = (\chi, \theta, \beta, \hat{\beta}, \varrho) \leftarrow_r (\mathbb{F}_p^*)^5$.

$\mathsf{K}_{\mathsf{pk}}(\mathsf{gk}, n, \mathsf{td})$: Let $\boldsymbol{P} = (P_i(\chi))_{i=1}^n$, $\hat{\boldsymbol{P}} = (\hat{P}_i(\theta))_{i=1}^n$, $\boldsymbol{Q} = ((P_i(\chi) + P_0(\chi))^2 - 1)_{i=1}^n$.
    $\mathsf{pk}_{uv} \leftarrow \begin{pmatrix} [1,\ P_0(\chi),\ \boldsymbol{P},\ \varrho,\ \boldsymbol{Q}/\varrho,\ \sum_{i=1}^n \hat{P}_i,\ \beta^2\varrho,\ \beta\hat{\beta},\ \beta^2\boldsymbol{P} + \beta\hat{\beta}\hat{\boldsymbol{P}}]_1, \\ [1,\ P_0(\chi),\ \boldsymbol{P},\ \varrho,\ \beta^2,\ \beta\hat{\beta}]_2,\ [1]_T \end{pmatrix}$ ,
    $\mathsf{pk}_{pkv} \leftarrow \begin{pmatrix} [\beta,\ \hat{\beta},\ (\theta^{2i-1})_{i=1}^n]_1, \\ [\chi, \theta, \beta, \hat{\beta}]_2 \end{pmatrix}$, $\mathsf{pk}_{vf} \leftarrow \begin{pmatrix} [(\chi^i)_{i=1}^{2n},\ (\beta\chi^i, \hat{\beta}\theta^{2i})_{i=1}^n, \beta\varrho]_1, \\ [(\chi^i)_{i=2}^n]_2 \end{pmatrix}$.
    Return $\mathsf{pk} \leftarrow ([\hat{\boldsymbol{P}}]_1, \mathsf{pk}_{uv}, \mathsf{pk}_{pkv}, \mathsf{pk}_{vf})$.

$\mathsf{K}(\mathsf{gk}, n)$: Run $\mathsf{td} \leftarrow \mathsf{K}_{\mathsf{td}}(\mathsf{gk})$, $\mathsf{pk} \leftarrow \mathsf{K}_{\mathsf{pk}}(\mathsf{gk}, n, \mathsf{td})$, return $(\mathsf{pk}, \mathsf{td})$.

$\mathsf{P}(\mathsf{gk}, (\mathsf{pk}_{\mathsf{e}}, \mathsf{pk}), [\boldsymbol{C}]_2 = [(\boldsymbol{c}_i)_{i=1}^n]_2 \in \mathbb{G}_2^{n \times 2}, (\sigma \in \mathbb{S}_n, \boldsymbol{t} \in \mathbb{F}_p^n))$:
    1. For $i = 1$ to $n - 1$: $\hat{r}_i \leftarrow_{\$} \mathbb{F}_p$; $[\hat{a}_i]_1 \leftarrow [\hat{P}_{\sigma^{-1}(i)}]_1 + \hat{r}_i[1]_1$.
    2. $\pi_{per} \leftarrow \mathsf{P}_{per}(\mathsf{gk}, \mathsf{pk}, [(\hat{a}_i)_{i=1}^{n-1}]_1, (\sigma, (\hat{r}_i)_{i=1}^{n-1}))$. // Permutation argument
    3. $\hat{r}_n \leftarrow -\sum_{i=1}^{n-1} \hat{r}_i$; $\hat{r} \leftarrow_r \mathbb{F}_p$; $[s]_1 \leftarrow \boldsymbol{t}^\top [\hat{\boldsymbol{P}}]_1 + \hat{r}[1]_1$.
    4. For $i = 1$ to $n$: $[\boldsymbol{t}_i']_2 \leftarrow t_i \cdot \mathsf{pk}_{\mathsf{e}}$.
    5. $[\boldsymbol{N}]_2 \leftarrow \hat{\boldsymbol{r}}^\top [\boldsymbol{C}]_2 + \hat{r} \cdot \mathsf{pk}_{\mathsf{e}}$. // Online
    6. $[\boldsymbol{C}']_2 \leftarrow ([\boldsymbol{c}_{\sigma(i)}]_2 + [\boldsymbol{t}_i']_2)_{i=1}^n$. // Shuffling, online
    7. Return $\big([\boldsymbol{C}']_2, \pi_{sh} \leftarrow ([(\hat{a}_j)_{j=1}^{n-1}, s]_1, [\boldsymbol{N}]_2, \pi_{per})\big)$.

$\mathsf{V}(\mathsf{gk}, (\mathsf{pk}_{\mathsf{e}}, \mathsf{pk}), ([\boldsymbol{C}]_2, [\boldsymbol{C}']_2), \pi_{sh})$:
    1. Parse $\pi_{sh} = ([(\hat{a}_j)_{j=1}^{n-1}, s]_1, [\boldsymbol{N}]_2, \pi_{per})$; set $[\hat{a}_n]_1 \leftarrow [\sum_{i=1}^n \hat{P}_i]_1 - \sum_{i=1}^{n-1} [\hat{a}_i]_1$.
    2. Check $\mathsf{V}_{per}(\mathsf{gk}, \mathsf{pk}, [(\hat{a}_i)_{i=1}^{n-1}]_1, \pi_{per}) = 1$.
    3. Check $[\hat{\boldsymbol{P}}]_1^\top \bullet [\boldsymbol{C}']_2 - [\hat{\boldsymbol{a}}]_1^\top \bullet [\boldsymbol{C}]_2 = [s]_1 \bullet \mathsf{pk}_{\mathsf{e}} - [1]_1 \bullet [\boldsymbol{N}]_2$.
```

Fig. 3. tFLSZ argument

```
$\mathsf{K}(\mathsf{gk}, n)$: Generate random $\mathsf{td} = (\chi, \beta, \hat{\beta}, \varrho, \hat{\varrho}, \mathsf{sk}_{\mathsf{e}}) \leftarrow_r (\mathbb{F}_p^*)^6$. Denote $\boldsymbol{P} = (P_i(\chi))_{i=1}^n$,
    $P_0 = P_0(\chi)$, and $\hat{\boldsymbol{P}} = (\hat{P}_i(\chi))_{i=1}^n$, $\boldsymbol{Q} = ((P_i + P_0)^2 - 1)_{i=1}^n$. Let
        $\mathsf{crs}_{sm} \leftarrow \big([\beta\boldsymbol{P} + \hat{\beta}\hat{\boldsymbol{P}}, \beta\varrho, \hat{\beta}\hat{\varrho}]_1, [\beta, \hat{\beta}]_2\big)$, $\mathsf{crs}_{con} \leftarrow [\begin{smallmatrix} \hat{\boldsymbol{P}} \\ \hat{\varrho} \end{smallmatrix}]_1$ , $\mathsf{pk}_{\mathsf{e}} = [1, \mathsf{sk}_{\mathsf{e}}]_2$
        $\mathsf{crs}_{pm} \leftarrow \big([1, P_0, \boldsymbol{Q}/\varrho, \sum_{i=1}^n P_i, \sum_{i=1}^n \hat{P}_i]_1, [P_0, \sum_{i=1}^n P_i]_2, [1]_T\big)$ .
    Set $\mathsf{crs} \leftarrow \big(\mathsf{pk}_{\mathsf{e}}, [\begin{smallmatrix} \boldsymbol{P} \\ \varrho \end{smallmatrix}]_1, [\begin{smallmatrix} \boldsymbol{P} \\ \varrho \end{smallmatrix}]_2, \mathsf{crs}_{sm}, \mathsf{crs}_{pm}, \mathsf{crs}_{con}\big)$. Return $(\mathsf{crs}, \mathsf{td})$.
```

Fig. 4. CRS generation algorithm of FLSZ

– Firstly, we need to add all the trapdoor elements to both groups which means adding $[\chi, \beta, \hat{\beta}]_1$ and $[\chi]_2$ to the public key.
– To evaluate polynomials $P_i(X)$ at point χ we add powers of χ in both groups to the public key. Since P_i is at most degree n, it suffices to include elements $[(\chi^i)_{i=1}^n]_1$ and $[(\chi^i)_{i=1}^n]_2$. However, since $(P_i(X) + P_0(X))^2 - 1$ has at most degree $2n$, we additionally add $[(\chi^i)_{i=n+1}^{2n}]_1$.

– Polynomials $\hat{P}_i$ have a degree $(i+1)(n+1)$, requiring, for the sake of verification friendliness, to include elements $[(\chi^i)_{i=1}^{(n+1)^2}]_1$ which would cause quadratic overhead. We avoid this by redefining the polynomials $\hat{P}_i$ and evaluating them at a new random point θ. The first idea would be to set $\hat{P}_i(X_\theta) = X_\theta^i$ for $i = 1, \ldots, n$ and add $[(\theta^i)_{i=1}^n]_1$ and $[\theta]_2$ to the public key. However, the $((\hat{P}_i(X_\theta))_{i=1}^n, 1)$-commitment scheme would not be binding since the KerMDH assumption does not hold for $[\boldsymbol{M}]_1 = [\hat{P}_1(X_\theta), \ldots, \hat{P}_n(X_\theta), 1]_1$, as the adversary can output $[\boldsymbol{c}]_2 = [\theta, -1, 0, \ldots, 0]_2$ such that $\boldsymbol{M}\boldsymbol{c}^\top = \boldsymbol{0}$ and $\boldsymbol{c} \neq \boldsymbol{0}$. Instead we set $\hat{P}_i(X_\theta) = X_\theta^{2i}$ for $i \in [1 \mathinner{..} n]$ and include $[(\theta^i)_{i=1}^{2n}]_1$ and $[\theta]_2$ to the public key. Now the commitment scheme is binding under a slight variation of the standard KerMDH assumption, which we prove in Section 5 to reduce to PDL assumption in the algebraic group model.

Another challenge is computing crs_{sm} since it contains elements $[\beta P_i + \hat{\beta}\hat{P}_i]_1$. It is not possible to reveal $[\beta P_i]_1$ and $[\hat{\beta}\hat{P}_i]_1$ since this breaks knowledge-soundness of the same-message argument. We propose a new argument to overcome this.

New Unit Vector Argument. We combine the same-message argument and unit vector argument from FLSZ to a new unit vector argument which is a proof of knowledge for the relation $\mathcal{R}_n^{uv} := \{([\hat{a}]_1, (I \in [1 \mathinner{..} n], \hat{r})) \mid \hat{a} = \hat{P}_I + \hat{r}\}$. The new argument in Fig. 5 has two advantages: (i) it has a verification-friendly public key, and (ii) the unit vector argument of FLSZ is sound only if we give a corresponding proof for the same-message argument; the new argument avoids this dependency. On a high level, the verification equation in Step 2 of V_{uv} and the proof element $[d]_1$ in Fig. 5 correspond to a variation of the same-message argument in FLSZ and shows that $[\hat{a}]_1$ and $[a]_1$ commit to the same message $\boldsymbol{m}$ respectively with the $((\hat{P}_i(X))_{i=1}^n, 1)$-commitment and the $((P_i(X))_{i=1}^n, X_\varrho)$-commitment. The verification equation in Step 3 of V_{uv} and elements $[b]_2$ and $[e]_1$ in Fig. 5 use the result of Lemma 1 to show that $[a]_1$ commits to a unit vector. This part is identical to the unit vector argument in FLSZ.

The main differences in the new argument are the public key elements for showing that $[\hat{a}]_1$ and $[a]_1$ commit to the same message. Simply revealing elements $[\beta P_i, \hat{\beta}\hat{P}_i]_1$ would be sufficient for verification-friendliness, but breaks the knowledge-soundness property: the same-message argument of FLSZ relies on $[\beta P_i(\chi) + \hat{\beta}\hat{P}_i(\theta)]_1$ being the only $\mathbb{G}_1$ elements in the span of $\{[\beta\chi^i + \hat{\beta}\theta^j]_1\}_{i,j}$ that are available to the adversary. Instead, we essentially substitute $[\beta P_i + \hat{\beta}\hat{P}_i]_1$ with $[\beta^2 P_i + \beta\hat{\beta}\hat{P}_i]_1$ (and other related elements accordingly), and equivalently use the fact that those are the only $\mathbb{G}_1$ elements in the span of $\{[\beta^2\chi^i + \beta\hat{\beta}\theta^j]_1\}_{i,j}$ available to the adversary. This change is significant since the latter elements can be computed with the DKG protocol without revealing $[\beta^2 P_i]_1$ and $[\beta\hat{\beta}\hat{P}_i]_1$:

(i) compute $[\beta\chi^i]_1$ and $[\hat{\beta}\theta^{2i}]_1 = [\hat{\beta}\hat{P}_i]_1$ to obtain $[\beta P_i + \hat{\beta}\hat{P}_i]_1$;
(ii) compute $[\beta^2 P_i + \beta\hat{\beta}\hat{P}_i]_1 = \mathsf{MD}_{\beta,1}([\beta P_i + \hat{\beta}\hat{P}_i]_1)$;
(iii) similarly, from elements $[\beta, \beta\varrho, \hat{\beta}]_1$ compute $[\beta^2\varrho]_1$ and $[\beta\hat{\beta}]_1$.

```
$\mathsf{K}_{uv}(\mathsf{gk}, n)$: Return $(\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{K}(\mathsf{gk}, n)$ from Fig. 3.
$\mathsf{P}_{uv}(\mathsf{gk}, \mathsf{pk}, [\hat{a}]_1), (I, \hat{r}))$:
    1. $r \leftarrow_\$ \mathbb{F}_p$, $[r']_1 \leftarrow r[\varrho]_1$, $[d]_1 \leftarrow [\beta^2 P_I + \beta\hat{\beta}\hat{P}_I]_1 + r[\beta^2\varrho]_1 + \hat{r}[\beta\hat{\beta}]_1$.
    2. $[a]_1 \leftarrow [P_I]_1 + [r']_1$, $[b]_2 \leftarrow [P_I]_2 + r[\varrho]_2$.
    3. $[e]_1 \leftarrow r \cdot (2([a]_1 + [P_0]_1) - [r']_1) + [\big((P_I + P_0)^2 - 1\big)/\varrho]_1$.
    4. Return $\pi_{uv} \leftarrow ([d]_1, [a]_1, [b]_2, [e]_1)$.
$\mathsf{V}_{uv}(\mathsf{gk}, \mathsf{pk}, [\hat{a}]_1, \pi_{uv})$:
    1. Parse $\pi_{uv} = ([d]_1, [a]_1, [b]_2, [e]_1)$ and pick $\alpha \leftarrow_\$ \mathbb{F}_p$.
    2. Check $[d]_1 \bullet [1]_2 = [a, \hat{a}]_1 \bullet [\beta^2, \beta\hat{\beta}]_2^\top$.
    3. Check $([a]_1 + \alpha[1]_1 + [P_0]_1) \bullet ([b]_2 - \alpha[1]_2 + [P_0]_2) = [e]_1 \bullet [\varrho]_2 + (1 - \alpha^2)[1]_T$.
```

Fig. 5. New unit vector argument

```
$\mathsf{K}_{per}(\mathsf{gk}, n)$: Return $(\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{K}(\mathsf{gk}, n)$ from Fig. 3.
$\mathsf{P}_{per}(\mathsf{gk}, \mathsf{pk}, [(\hat{a}_i)_{i=1}^{n-1}]_1, (\sigma \in \mathbb{S}_n, (\hat{r}_i)_{i=1}^{n-1}))$:
    1. $\hat{r}_n \leftarrow -\sum_{i=1}^{n-1} \hat{r}_i$, $[\hat{a}_n]_1 \leftarrow [\sum_{i=1}^{n} \hat{P}_i]_1 - \sum_{i=1}^{n-1} [\hat{a}_i]_1$.
    2. For $i \in [1 \mathinner{..} n]$: $\pi_{uv:i} \leftarrow \mathsf{P}_{uv}(\mathsf{gk}, \mathsf{pk}, [\hat{a}_i]_1, (\sigma^{-1}(i), \hat{r}_i))$.
    3. Return $\pi_{per} \leftarrow (\pi_{uv:i})_{i=1}^{n}$.
$\mathsf{V}_{per}(\mathsf{gk}, \mathsf{pk}, [(\hat{a}_i)_{i=1}^{n-1}]_1, \pi_{per})$:
    1. Parse $\pi_{per} = (\pi_{uv:i})_{i=1}^{n}$ and set $[\hat{a}_n]_1 \leftarrow [\sum_{i=1}^{n} \hat{P}_i]_1 - \sum_{i=1}^{n-1} [\hat{a}_i]_1$.
    2. For $i \in [1 \mathinner{..} n]$ : check $\mathsf{V}_{uv}(\mathsf{gk}, \mathsf{pk}, [\hat{a}_i]_1, \pi_{uv:i}) = 1$.
```

Fig. 6. Permutation argument

Additionally, in $\mathbb{G}_2$ we reveal $[\beta^2]_2$ and $[\beta\hat{\beta}]_2$. We prove in Appendix C that these changes retain security.

Permutation Argument. The permutation argument is a proof of knowledge for the relation

$$\mathcal{R}_{per} = \left\{ ([\hat{\boldsymbol{a}}]_1, (\sigma, \hat{\boldsymbol{r}})) \mid \sigma \in \mathbb{S}_n \wedge \textstyle\sum_{i=1}^{n} \hat{r}_i = 0 \wedge (\forall i \in [1 \mathinner{..} n] : \hat{a}_i = \hat{P}_{\sigma^{-1}(i)} + \hat{r}_i) \right\} .$$

We show that this relation is fulfilled the same way as previous NIZK shuffle arguments. Firstly, the prover gives a unit vector argument for each of the commitments $[\hat{a}_i]_1$ for $i \in [1 \mathinner{..} n-1]$. Next, observe that only if those commitments are to distinct values $\hat{P}_i$, is $[\hat{a}_n]_1 := [\sum_{i=1}^{n} \hat{P}_i]_1 - \sum_{i=1}^{n-1} [\hat{a}_i]_1$ a unit vector. Hence, by giving a unit vector argument also for $[\hat{a}_n]_1$, where $[\hat{a}_n]_1$ is explicitly computed by the verifier, we have proven the relation. Condition $\sum_{i=1}^{n} \hat{r}_i = 0$ in $\mathcal{R}_{per}$ comes from the way that $[\hat{a}_n]_1$ is computed. The protocol is given in Fig. 6.

Shuffle Argument. Finally, we prove that ciphertexts were shuffled according to the permutation σ committed in $[\hat{\boldsymbol{a}}]_1$. This is essentially equivalent to the consistency argument in FLSZ. Intuitively, we check that $\sum_{i=1}^{n} [\hat{P}_i]_1 \bullet [m'_i]_2 = \sum_{i=1}^{n} [\hat{P}_{\sigma^{-1}(i)}]_1 \bullet [m_i]_2$ (see Step 3 for the actual equation) which guarantees that $\sum_{i=1}^{n} [\hat{P}_i]_1 \bullet ([m'_i]_2 - [m_{\sigma(i)}]_2) = [0]_T$. If $[m'_i]_2 \neq [m_{\sigma(i)}]_2$ for some i, then

```
Input: $(\chi, \theta, \beta, \hat{\beta}, \varrho) \in (\mathbb{F}_p^*)^5$.
Layer $C_1$:
  1. For $\iota \in \{\beta, \hat{\beta}, \varrho\}$, $z \in \{1, 2\}$: $[\iota]_z \leftarrow \iota\,[1]_z$.
  2. For $i = 1$ to $2n$: $[\chi^i]_1 \leftarrow \chi[\chi^{i-1}]_1$ ,  $[\theta^i]_1 \leftarrow \theta[\theta^{i-1}]_1$.
  3. Set $[\theta]_2 \leftarrow \theta[1]_2$ and denote $[\hat{P}_i]_1 = [\theta^{2i}]_1$ for $i = 1$ to $n$.
  4. For $i = 1$ to $n$: $[\chi^i]_2 \leftarrow \chi[\chi^{i-1}]_2$, $[\beta\chi^i]_1 \leftarrow \beta[\chi^i]_1$ ,  $[\hat{\beta}\hat{P}_i]_1 \leftarrow \hat{\beta}[\hat{P}_i]_1$.
  5. $[\beta^2]_2 \leftarrow \beta[\beta]_2$, $[\beta\varrho]_1 \leftarrow \beta[\varrho]_1$, $[\beta^2\varrho]_1 \leftarrow \beta[\beta\varrho]_1$, $[\beta\hat{\beta}]_1 \leftarrow \beta[\hat{\beta}]_1$, $[\beta\hat{\beta}]_2 \leftarrow \beta[\hat{\beta}]_2$.
Layer $L_1$:
  1. Compute $[(\ell_i(\chi),\ \beta\ell_i(\chi),\ \ell_i(\chi)^2)_{i=1}^{n+1}]_1$, $[(\ell_i(\chi))_{i=1}^{n+1}]_2$, $[(\ell_i(\chi) \cdot \ell_{n+1}(\chi))_{i=1}^{n}]_1$ from
     $[(\chi^i)_{i=0}^{2n}]_1$, $[(\beta\chi^i)_{i=0}^{n}]_1$, and $[(\chi^i)_{i=0}^{n}]_2$ as shown in Appendix A.
  2. $[P_0]_1 \leftarrow [\ell_{n+1}(\chi)]_1 - [1]_1$ ,  $[P_0]_2 \leftarrow [\ell_{n+1}(\chi)]_2 - [1]_2$.
  3. For $i = 1$ to $n$:
     (a) $[P_i]_1 \leftarrow 2[\ell_i(\chi)]_1 + [\ell_{n+1}(\chi)]_1$, $[P_i]_2 \leftarrow 2[\ell_i(\chi)]_2 + [\ell_{n+1}(\chi)]_2$
     (b) $[Q_i]_1 \leftarrow 4[\ell_i(\chi)^2]_1 + 4[\ell_{n+1}(\chi)^2]_1 + 8[\ell_i(\chi) \cdot \ell_{n+1}(\chi)]_1 - 4[\ell_i(\chi)]_1 - 4[\ell_{n+1}(\chi)]_1$.
     (c) $[\beta P_i + \hat{\beta}\hat{P}_i]_1 \leftarrow 2[\beta\ell_i(\chi)]_1 + [\beta\ell_{n+1}(\chi)]_1 + [\hat{\beta}\theta^{2i}]_1$.
  4. $[\sum_{i=1}^{n} \hat{P}_i]_1 \leftarrow \sum_{i=1}[\hat{P}_i]_1$.
Layer $C_2$: For $i = 1$ to $n$: $[Q_i/\varrho]_1 \leftarrow [Q_i]_1/\varrho$ ,  $[\beta^2 P_i + \beta\hat{\beta}\hat{P}_i]_1 \leftarrow \beta[\beta P_i + \hat{\beta}\hat{P}_i]_1$.
```

Fig. 7. Public key computation as a circuit

the adversary can find a non-zero element in the kernel of $[\hat{\boldsymbol{P}}]_1$ and thus break the KerMDH assumption. Of course, the actual messages m_i are encrypted and the verifier knows only a commitment to σ. We balance this in the equation by allowing the prover to produce elements $[s]_1$ and $[\boldsymbol{N}]_2$, which cancels the randomness in the ciphertexts and the commitments.

Verification-Friendliness of tFLSZ*.* After making all of the above modifications we end up with a public key as presented in Fig. 3. There are two new sub-keys: pk_{pkv} which contains some elements later required by the V_{pk} algorithm (used by prover to guarantee nn-ZK), and pk_{vf} which is a by-product of making the public key verification-friendly. After the public key generation protocol has finished the elements in pk_{vf} can be disregarded. It is now simple to verify that the public key is verification-friendly. We present it as a series of multiplication-division and linear combination layers in Fig. 7. Hence, the DKG protocol described in Section 3 can be applied. For the sake of completeness, we provide an explicit description of the DKG protocol in Appendix B.

For better modularity, we treat the encryption key pk_{e} separately from the argument's public key. However, we assume it to be correctly generated by some secure DKG protocol, such as the one by Gennaro et al. [GJKR99].

Theorem 1 ([ABL⁺19b]). *If* tFLSZ *is complete, sound, and computational zero-knowledge in the CRS model, then it is complete, sound, and computational zero-knowledge if the adversary corrupts all but one party in the DKG protocol.*

$\mathsf{Sim}_{per}(\mathsf{gk}, \mathsf{pk}, (\beta, \hat{\beta}), [(\hat{a})_{i=1}^{n-1}]_1)$: Set $[\hat{a}_n]_1 \leftarrow [\sum_{i=1}^{n} \hat{P}_i]_1 - \sum_{i=1}^{n-1}[\hat{a}_i]_1$, compute $\pi_{uv:i} \leftarrow \mathsf{Sim}_{uv}(\mathsf{gk}, [\hat{a}_i]_1, (\beta, \hat{\beta}))$ for $i \in [1 \mathinner{..} n]$, and return $\pi_{per} \leftarrow (\pi_{uv:i})_{i=1}^{n}$.

Fig. 8. Simulator of the permutation argument

5 Security in the CRS Model

In this section, we establish that tFLSZ is secure in the CRS model, where the CRS is the public key generated by a trusted party. We first claim security of the unit vector and permutation arguments, as stated in Theorem 2 and Theorem 3.

Theorem 2 (Security of unit vector argument). *The unit vector argument in the CRS model (see Fig. 5) has perfect completeness and perfect zero-knowledge. If the* $(3n-1)$*-PDL assumption holds, then it has computational knowledge-soundness in the AGM.*

The proof of Theorem 2 is given in Appendix C. Soundness of the unit vector argument uses a common trick of AGM proofs that first defines an *idealised verification*, where the verification equation holds true for polynomials $V(\boldsymbol{X})$ (with trapdoor elements as variables) rather than for polynomial evaluations $V(\boldsymbol{\chi})$ only (*real verification*, for concrete trapdoor elements $\boldsymbol{\chi}$). We then show that no element outside the unit vector language can pass the idealised verification. Moreover, if an adversary manages to pass the real verification but not the ideal one, then she can be used to break the $(3n-1)$-PDL assumption. The proof of the other properties are quite standard.

Theorem 3 (Security of permutation argument). *The permutation argument in the CRS model (see Fig. 6) is perfectly complete and perfectly zero-knowledge. If the unit vector argument is knowledge-sound and* $((\hat{P}_i(X))_{i=1}^{n}, 1)$*-commitment is binding, then the permutation argument is also knowledge-sound.*

Proof. Perfect completeness. Observe that $[\hat{a}_n]_1 = [\sum_{i=1}^{n} \hat{P}_i]_1 - \sum_{i=1}^{n-1}[\hat{a}_i]_1 = [\sum_{i=1}^{n} \hat{P}_i]_1 - \sum_{i=1}^{n-1}[\hat{P}_{\sigma^{-1}(i)} + \hat{r}_i]_1 = [\hat{P}_{\sigma^{-1}(n)} + \hat{r}_n]_1$. Therefore, completeness follows from completeness of the unit vector argument.

Perfect zero-knowledge. Observe that P_{per} and Sim_{per} compute $[\hat{a}_n]_1$ the same way and the proof is just n unit vector arguments which we have already shown to have perfect zero-knowledge.

Knowledge-soundness. Let $\mathcal{A}_{per}$ be a PPT adversary against knowledge-soundness. According to the unit vector argument, for each $i \in [1 \mathinner{..} n]$, there exists an extractor $\mathsf{Ext}_{uv:i}$ such that $\big(([(\hat{a})_{i=1}^{n-1}]_1, \pi_{per}) \| (I_i, \hat{r}_i)\big) \leftarrow (\mathcal{A}_{per} \| \mathsf{Ext}_{uv:i})(\mathsf{gk}, \mathsf{pk})$ where the probability that $\mathsf{V}_{per}(\mathsf{gk}, [(\hat{a})_{i=1}^{n-1}]_1, \pi_{per}) = 1$

and $\hat{a}_i \neq \hat{P}_{I_i} + \hat{r}_i$ is negligible. Hence, if verification accepts, with overwhelming probability $\hat{a}_i = \hat{P}_{I_i} + \hat{r}_i$. On the other hand $\hat{a}_n = \sum_{i=1}^{n} \hat{P}_i - \sum_{j=1}^{n-1} \hat{a}_j = \sum_{i=1}^{n} \hat{P}_i - \sum_{j=1}^{n-1} \left(\hat{P}_{I_j} + \hat{r}_j\right)$. Considering that $\hat{P}_{I_j}$ for $j \in [1 \mathinner{..} n-1]$ might not be distinct, we can express $\hat{a}_n = \sum_{i=1}^{n} \hat{P}_i - \sum_{j=1}^{n} k_j \hat{P}_j + (-\sum_{i=j}^{n-1} \hat{r}_j) = \sum_{i=1}^{n} (1-k_i)\hat{P}_i + (-\sum_{j=1}^{n-1} \hat{r}_j)$ where $k_i \in [0 \mathinner{..} n-1]$. Since also $\hat{a}_n = \hat{P}_{I_n} + \hat{r}_n$, we must have $(1-k_i) = 0$ for $i \in [1 \mathinner{..} n] \setminus \{I_n\}$, $k_{I_n} = 0$, and $\hat{r}_n = -\sum_{j=1}^{n-1} \hat{r}_j$ since otherwise $(1-k_1, \dots, 1-k_n, -\sum_{j=1}^{n-1} \hat{r}_j)$ and $(\boldsymbol{e_{I_n}}, \hat{r}_n)$, where $\boldsymbol{e_{I_n}}$ is the I_n-th unit vector, are different openings for the commitment $[\hat{a}_n]_1$. It follows that $P_{I_1}, \dots, P_{I_n}$ are distinct, so we can find a unique permutation $\sigma \in \mathbb{S}_n$ such that $P_{I_i} = P_{\sigma^{-1}(i)}$ for $i \in [1 \mathinner{..} n]$. The extractor Ext_{per}, that runs $\mathsf{Ext}_{uv:i}$ for $i \in [1 \mathinner{..} n]$, outputs $(\sigma, (\hat{r}_i)_{i=1}^{n-1})$ found in such a way. □

We prove soundness of the shuffle argument under a weaker assumption compared to [FLSZ17a]. The assumption, called the GapKerMDH assumption, is novel, but we show that it reduces to the PDL assumption in the AGM. More precisely, since the KerMDH assumption is insecure for $M = (1, \theta, \dots, \theta^n) \in \mathbb{Z}_p^{n \times 1}$ if the adversary is given both $[M]_1$ and $[\theta]_2$, then a slightly modified assumption is required. We still give the same information to the adversary, but require that the output is in the kernel of a certain $M' \subset M$ that contains periodic gaps.

Definition 9. *The n-GapKerMDH assumption holds for* BGen *if for any PPT $\mathcal{A}$,*

$$\Pr\begin{bmatrix}\mathsf{gk} \leftarrow \mathsf{BGen}(1^\lambda), \theta \leftarrow_\$ \mathbb{F}_p^*, [\boldsymbol{v}]_2 \leftarrow \mathcal{A}(\mathsf{gk}, [(\theta^i)_{i=1}^{2n}]_1, [\theta]_2) : \\ \boldsymbol{v}^\top \cdot (\theta^{2i})_{i=0}^{n} = 0 \wedge \boldsymbol{v} \neq \boldsymbol{0}_{n+1}\end{bmatrix} \approx_\lambda 0.$$

Theorem 4. *If the $(2n)$-PDL assumption holds, then the n-GapKerMDH assumption holds in the AGM.*

Proof. Let $\mathcal{A}$ be an algebraic PPT adversary that breaks n-GapKerMDH assumption with probability ε_{gap}. More precisely, $\mathcal{A}$ gets as an input $(\mathsf{gk}, [(\theta^i)_{i=1}^{2n}]_1, [\theta]_2)$ for $\theta \leftarrow_\$ \mathbb{Z}_p$, and outputs a non-zero $[\boldsymbol{v}]_2 \in \mathbb{G}_2^{n+1}$ and its linear representation $\boldsymbol{U} \in \mathbb{Z}_p^{(n+1)\times 2}$ (that is $[\boldsymbol{v}]_2 = \boldsymbol{U} \cdot [1, \theta]_2^\top$) such that $\sum_{i=0}^{n} \theta^{2i} \cdot v_{i+1} = 0$.

We construct a PPT adversary $\mathcal{B}$ that breaks $(2n)$-PDL assumption using $\mathcal{A}$. First, $\mathcal{B}$ gets as an input $(\mathsf{gk}, [(\theta^i)_{i=1}^{2n}]_1, [(\theta^i)_{i=1}^{2n}]_2)$ and runs $\mathcal{A}(\mathsf{gk}, [(\theta^i)_{i=1}^{2n}]_1, [\theta]_2)$ to get the output $[\boldsymbol{v}]_2$ and $\boldsymbol{U}$. Let us define polynomials $V_i(X_\theta) := U_{i,1} + U_{i,2} \cdot X_\theta$ for $i \in [1 \mathinner{..} n+1]$ which in particular satisfies $V_i(\theta) = v_i$. Similarly for the expression $\sum_{i=0}^{n} \theta^{2i} \cdot v_{i+1}$ we define a polynomial $V(X_\theta) := \sum_{i=0}^{n} X_\theta^{2i} \cdot V_{i+1}(X_\theta)$ such that if $\mathcal{A}$ wins then $V(\theta) = 0$. Adversary $\mathcal{B}$ will abort if $\mathcal{A}$ either outputs an incorrect representation U or loses the n-GapKerMDH game. Otherwise $\mathcal{B}$ finds roots of $V(X_\theta)$ (can be done efficiently), and returns the one which matches $[\theta]_1$.

Finding roots of $V(X_\theta)$ is only possible if $V(X_\theta)$ is a non-zero polynomial, but it is easy to see that this is always the case if $\mathcal{A}$ wins. We may express

$$V(X_\theta) = \sum_{i=0}^{n} X_\theta^{2i} \cdot (U_{i+1,1} + U_{i+1,2} \cdot X_\theta) = \sum_{i=0}^{n} U_{i+1,1} X_\theta^{2i} + \sum_{i=0}^{n} U_{i+1,2} \cdot X_\theta^{2i+1} .$$

So if $V(X_\theta) = 0$ then $\boldsymbol{U} = 0$ and therefore $\boldsymbol{v} = 0$ which contradicts $\mathcal{A}$ winning. It follows that $\mathcal{B}$ can break the $(2n)$-PDL assumption with probability ε_{gap}. $\square$

Theorem 5 (Security of shuffle argument). tFLSZ *is perfectly complete and perfectly zero-knowledge in the CRS model. If the permutation argument is knowledge-sound and the n-GapKerMDH assumption holds, then* tFLSZ *is sound.*

Proof. Perfect completeness. Can be straightforwardly verified by substituting an honest proof to the verification equations.

Perfect zero-knowledge. We show that the simulator Sim in Fig. 9 outputs an argument that has the same distribution as an argument output by an honest prover. In both cases $[(a_i)_{i=1}^n]_1$, $[(\hat{a}_i)_{i=1}^{n-1}]_1$, and $[s]_1$ are uniformly randomly and independently distributed group elements. Moreover, both honest and simulated arguments have $b_i = a_i$ for $i \in [1 \mathinner{..} n]$ and $[\hat{a}_n]_1 = \sum_{i=1}^n [\hat{P}_i]_1 - \sum_{i=1}^{n-1} [\hat{a}_i]_1$. Elements $[\boldsymbol{d}]_1$, $[\boldsymbol{e}]_1$, $[\boldsymbol{N}]_2$ are now uniquely fixed by the verification equation and the elements mentioned before. It is straightforward to check that the simulated argument satisfies the verification equations. Thus the distributions are equal.

Soundness. Let $\mathcal{A}_{sh}$ be a PPT adversary that breaks soundness of the shuffle argument with probability ε_{sh}. Let $\mathcal{A}_{per}$ be the adversary $\mathcal{A}_{sh}$ restricted only to output $([(\hat{a}_i)_{i=1}^{n-1}]_1, \pi_{per})$ and $\mathsf{Ext}_{\mathcal{A}_{per}}$ be an arbitrary extractor such that $\mathcal{A}_{per}$ breaks knowledge-soundness of the permutation argument with probability ε_{per}.

We construct an adversary $\mathcal{A}_{gap}$ against the n-GapKerMDH assumption that on input $(\mathsf{gk}, [(\theta^i)_{i=1}^{2n}]_1, [\theta]_2)$ proceeds as follows:

1. Sample $\chi, \beta, \hat{\beta}, \varrho \leftarrow (\mathbb{F}_p^*)^4$ and $\mathsf{sk_e} \leftarrow_\$ \mathbb{F}_p$. Set $\mathsf{pk_e} \leftarrow [1, \mathsf{sk_e}]$.
2. Compute pk using $[(\theta^i)_{i=1}^{2n}]_1$, $[\theta]_2$, and $\chi, \beta, \hat{\beta}, \varrho$. In particular, notice that $[\beta P_i(\chi) + \hat{\beta}\hat{P}_i(\theta)]_1 = (\beta P_i(\chi)) \cdot [1]_1 + \hat{\beta} \cdot [\theta^{2i}]_1$ and $[\hat{\beta}\theta^{2i}]_1 = \hat{\beta} \cdot [\theta^{2i}]_1$.
3. Sample $r_{sh} \leftarrow_\$ \mathsf{RND}(\mathcal{A}_{sh})$ and run $([\boldsymbol{C}, \boldsymbol{C}']_2, \pi_{sh}) \leftarrow \mathcal{A}_{sh}(\mathsf{gk}, (\mathsf{pk_e}, \mathsf{pk}); r_{sh})$.
4. If $\mathsf{V}(\mathsf{gk}, (\mathsf{pk_e}, \mathsf{pk}), ([\boldsymbol{C}]_2, [\boldsymbol{C}']_2), \pi_{sh}) \neq 1$, then abort.
5. Parse $\pi_{sh} = ([(\hat{a}_j)_{j=1}^{n-1}]_1, \pi_{per}, \pi_{con})$ and set $[\hat{a}_n]_1 \leftarrow [\sum_{i=1}^n \hat{P}_i]_1 - \sum_{i=1}^{n-1} [\hat{a}_i]_1$.
6. Run $(\sigma, \hat{\boldsymbol{r}}) \leftarrow \mathsf{Ext}_{\mathcal{A}_{per}}(\mathsf{gk}, \mathsf{pk}; r_{sh})$.
7. If $([\hat{\boldsymbol{a}}]_1, (\sigma, \hat{\boldsymbol{r}})) \notin \mathcal{R}_{per}$, then abort.
8. Set $\boldsymbol{A} \in \{0,1\}^{n \times n}$ such that $A_{i,j} = 1$ iff $\sigma^{-1}(i) = j$.
9. Set $[\boldsymbol{m}]_2 \leftarrow \mathsf{Dec}_{\mathsf{sk_e}}([\boldsymbol{C}]_2)$, $[\boldsymbol{m}']_2 \leftarrow \mathsf{Dec}_{\mathsf{sk_e}}([\boldsymbol{C}']_2)$, and $[z]_2 \leftarrow \mathsf{Dec}_{\mathsf{sk_e}}([\boldsymbol{N}]_2)$.
10. Return $[\boldsymbol{v}]_2 \leftarrow \begin{pmatrix} [\boldsymbol{m}']_2 - \boldsymbol{A}[\boldsymbol{m}]_2 \\ [z]_2 - \hat{\boldsymbol{r}}^\top [\boldsymbol{m}]_2 \end{pmatrix}$.

Let us analyse the success probability of $\mathcal{A}_{gap}$. Let X be the event that $\mathcal{A}_{sh}$ wins, i.e., there is no abort on step 4, and for any permutation matrix $\boldsymbol{P}$, we have

```
$\mathsf{Sim}(\mathsf{gk}, (\mathsf{pk_e}, \mathsf{pk}), \mathsf{td}, ([\boldsymbol{C}]_2, [\boldsymbol{C}']_2))$:
 1. For $i = 1$ to $n-1$: // commits to the identity permutation
    (a) $r_i, \hat{r}_i \leftarrow_\$ \mathbb{F}_p$;
    (b) $[a_i]_1 \leftarrow [P_i]_1 + r_i[\varrho]_1$; $[b_i]_2 \leftarrow [P_i]_2 + r_i[\varrho]_2$; $[\hat{a}_i]_1 \leftarrow [\hat{P}_i]_1 + \hat{r}_i[1]_1$;
 2. $r_n \leftarrow_\$ \mathbb{F}_p$; $\hat{r}_n \leftarrow -\sum_{i=1}^{n-1} \hat{r}_i$;
 3. $[a_n]_1 \leftarrow [P_n]_1 + r_n[\varrho]_1$; $[b_n]_2 \leftarrow [P_n]_2 + r_n[\varrho]_2$; $[\hat{a}_n]_1 \leftarrow \sum_{i=1}^{n}[\hat{P}_i]_1 - \sum_{i=1}^{n-1}[\hat{a}_i]_1$;
 4. For $i = 1$ to $n$:
    (a) $[d_i]_1 \leftarrow [\beta^2 P_i + \beta\hat{\beta}\hat{P}_i]_1 + r_i[\beta^2\varrho]_1 + \hat{r}_i[\beta\hat{\beta}]_1$;
    (b) $[e_i]_1 \leftarrow r_i \cdot (2([a_i]_1 + [P_0]_1) - r_i[\varrho]_1) + [Q_i/\varrho]_1$;
 5. $\hat{r} \leftarrow_\$ \mathbb{F}_p$; $[s]_1 \leftarrow \boldsymbol{0}^\top[\hat{\boldsymbol{P}}]_1 + \hat{r}[1]_1$; $[\boldsymbol{N}]_2 \leftarrow (\hat{\boldsymbol{P}} + \hat{\boldsymbol{r}})[\boldsymbol{C}]_2 - \hat{\boldsymbol{P}}[\boldsymbol{C}']_2 + \hat{r} \cdot \mathsf{pk_e}$;
 6. $\pi_{per} \leftarrow ([\boldsymbol{d}]_1, [\boldsymbol{a}]_1, [\boldsymbol{b}]_2, [\boldsymbol{e}]_1)$;
 7. Return $\pi_{sh} \leftarrow ([(\hat{a}_i)_{i=1}^{n-1}, s]_1, [\boldsymbol{N}]_2, \pi_{per})$.
```

Fig. 9. Simulator of tFLSZ

$[\boldsymbol{m}']_2 \neq \boldsymbol{P}[\boldsymbol{m}]_2$. Let Y be the event that $\mathcal{A}_{per}$ wins, i.e., $([\hat{\boldsymbol{a}}]_1, (\sigma, \hat{\boldsymbol{r}})) \notin \mathcal{R}_{per}$. Firstly, consider the case that X happens and Y does not happen. Then in particular: (i) $\mathcal{A}_{sh}$ does not abort, (ii) $\boldsymbol{A}$ is a permutation matrix that satisfies $[\hat{\boldsymbol{a}}]_1 = \begin{pmatrix} \boldsymbol{A} \\ \hat{\boldsymbol{r}}^\top \end{pmatrix}^\top \begin{bmatrix} \hat{\boldsymbol{P}} \\ 1 \end{bmatrix}_1$, (iii) $[\boldsymbol{m}']_2 \neq \boldsymbol{A}[\boldsymbol{m}]_2$, and (iv) the verification equation $[\hat{\boldsymbol{P}}]_1^\top \bullet [\boldsymbol{C}']_2 - [\hat{\boldsymbol{a}}]_1^\top \bullet [\boldsymbol{C}]_2 = [s]_1 \bullet \mathsf{pk_e} - [1]_1 \bullet [\boldsymbol{N}]_2$ is satisfied. By decrypting the ciphertexts in the last equation, we get

$$\begin{aligned}
[0]_T &= [\hat{\boldsymbol{P}}]_1^\top \bullet [\boldsymbol{m}']_2 - [\hat{\boldsymbol{a}}]_1^\top \bullet [\boldsymbol{m}]_2 + [1]_1 \bullet [z]_2 \\
&= [\hat{\boldsymbol{P}}]_1^\top \bullet [\boldsymbol{m}']_2 - \begin{bmatrix} \hat{\boldsymbol{P}} \\ 1 \end{bmatrix}_1^\top \begin{pmatrix} \boldsymbol{A} \\ \hat{\boldsymbol{r}}^\top \end{pmatrix} \bullet [\boldsymbol{m}]_2 + [1]_1 \bullet [z]_2 \\
&= [\hat{\boldsymbol{P}}]_1^\top \bullet [\boldsymbol{m}' - \boldsymbol{A}\boldsymbol{m}]_2 + [1]_1 \bullet [z - \hat{\boldsymbol{r}}^\top \boldsymbol{m}]_2 \\
&= \begin{bmatrix} \hat{\boldsymbol{P}} \\ 1 \end{bmatrix}_1^\top \bullet \begin{pmatrix} [\boldsymbol{m}']_2 - \boldsymbol{A}[\boldsymbol{m}]_2 \\ [z]_2 - \hat{\boldsymbol{r}}^\top [\boldsymbol{m}]_2 \end{pmatrix} = \begin{bmatrix} \hat{\boldsymbol{P}} \\ 1 \end{bmatrix}_1^\top \bullet [\boldsymbol{v}]_2 .
\end{aligned}$$

Since $[\boldsymbol{m}']_2 \neq \boldsymbol{A}[\boldsymbol{m}]_2$, then $[\boldsymbol{v}]_2 \neq [\boldsymbol{0}_{n+1}]_2$ is a solution to the n-GapKerMDH problem. Finally, we can express the success probability of $\mathcal{A}_{sh}$ as follows:

$$\varepsilon_{sh} = \Pr[X] = \Pr[X \wedge Y] + \Pr[X \wedge \neg Y] \leq \Pr[Y] + \Pr[X \wedge \neg Y] \leq \varepsilon_{per} + \varepsilon_{gap} .$$

Since there exists an extractor $\mathsf{Ext}_{\mathcal{A}_{per}}$ such that $\varepsilon_{per} \approx_\lambda 0$, it follows that $\varepsilon_{sh} \leq \varepsilon_{per} + \varepsilon_{gap} \approx_\lambda 0$. □

6 Zero-Knowledge in the BPK Model

We augment the prover in the BPK model with a key verification algorithm $\mathsf{V_{pk}}$ in Fig. 10 such that she outputs a proof only if the verification passes. Then we prove that tFLSZ is nn-ZK in the BPK model with respect to the $\mathsf{V_{pk}}$ algorithm. Firstly, we show that each subverter that creates a valid public key (one that is accepted by $\mathsf{V_{pk}}$) will know the trapdoors. Let $[\mathsf{td}']_1$ denote the vector in pk that is supposedly $[\chi, \theta, \beta, \hat{\beta}, \varrho]_1$.

```
$\mathsf{V_{pk}}(\mathsf{gk}, \mathsf{pk})$ :
 1. Check that $\mathsf{pk}$ can be parsed as in Fig. 3 and that each element belongs to the
    correct group.
 2. Check that $[\varrho]_1 \neq [0]_1$.
 3. Check that $[\iota]_1 \bullet [1]_2 = [1]_1 \bullet [\iota]_2$ for $\iota \in \{\chi, \theta, \beta, \hat{\beta}, \varrho\}$.
 4. Check that $[1]_T = [1]_1 \bullet [1]_2$.
 5. For $i = 2$ to $2n$: check that $[\theta^i]_1 \bullet [1]_2 = [\theta^{i-1}]_1 \bullet [\theta]_2$. // Note that $\hat{P}_i = \theta^{2i}$
 6. Check that $[1]_1 \bullet [\beta^2]_2 = [\beta]_1 \bullet [\beta]_2$.
 7. Check that $[\beta^2\varrho]_1 \bullet [1]_2 = [\varrho]_1 \bullet [\beta^2]_2$.
 8. Check that $[\beta\hat{\beta}]_1 \bullet [1]_2 = [\beta]_1 \bullet [\hat{\beta}]_2$.
 9. Check that $[1]_1 \bullet [\beta\hat{\beta}]_2 = [\beta\hat{\beta}]_1 \bullet [1]_2$.
10. Check that $[1]_1 \bullet [P_0]_2 = [P_0]_1 \bullet [1]_2$.
11. For $i = 1$ to $n$: check that
    (a) $[1]_1 \bullet [P_i]_2 = [P_i]_1 \bullet [1]_2$,
    (b) $[\beta^2 P_i + \beta\hat{\beta}\hat{P}_i]_1 \bullet [1]_2 = [P_i]_1 \bullet [\beta^2]_2 + [\hat{P}_i]_1 \bullet [\beta\hat{\beta}]_2$,
    (c) $[((P_i + P_0)^2 - 1)/\varrho]_1 \bullet [\varrho]_2 = ([P_i + P_0]_1 \bullet [P_i + P_0]_2) - [1]_T$.
```

Fig. 10. The $\mathsf{V_{pk}}$ algorithm of tFLSZ. For ease of presentation, the algorithm is described as if the public key was already well-formed.

Lemma 2. *Consider* $\mathsf{V_{pk}}$ *in Fig. 10 and suppose the BDH-KE assumption holds. Then, for any PPT subverter* X*, there exist a PPT extractor* $\mathsf{Ext_X}$ *such that,*

$$\Pr\left[(\mathsf{pk}, \mathsf{aux_X} \| \mathsf{td}) \leftarrow (\mathsf{X} \| \mathsf{Ext_X})(\mathsf{gk}) : \mathsf{V_{pk}}(\mathsf{gk}, \mathsf{pk}) = 1 \land [\mathsf{td}]_1 \neq [\mathsf{td}']_1 \subset \mathsf{pk}\right] \approx_\lambda 0.$$

Proof. The proof is similar to Theorem 4 in [ABLZ17]. If $\mathsf{V_{pk}}(\mathsf{gk}, \mathsf{pk}) = 1$, then: (i) Since Step 1 in $\mathsf{V_{pk}}$ is satisfied, there exist elements $[\mathsf{td}']_1 = [\chi', \theta', \beta', \hat{\beta}', \varrho']_1$ and $[\mathsf{td}'']_2 = [\chi'', \theta'', \beta'', \hat{\beta}'', \varrho'']_2$ in pk that supposedly correspond to trapdoor elements. (ii) By Step 3 $[\iota']_1 \bullet [1]_2 = [1]_1 \bullet [\iota'']_2$ and therefore $\iota' = \iota''$, for $\iota \in \{\chi, \theta, \beta, \hat{\beta}, \varrho\}$. According to BDH-KE, there exists an extractor Ext_ι that outputs ι' with overwhelming probability on the same random coins as X. Therefore, we can construct $\mathsf{Ext_X}(r)$ by simply returning $(\mathsf{Ext}_\iota(r))_{\iota \in \mathsf{td}}$. □

Theorem 6. *If BDH-KE assumption holds, then* tFLSZ *has statistical nn-ZK.*

Proof. From Lemma 2, we know that for any PPT X, there exists an extractor $\mathsf{Ext_X}$ that with overwhelming probability outputs the trapdoor td given that $\mathsf{V_{pk}}(\mathsf{gk}, \mathsf{pk}) = 1$. Let us show that if $\mathsf{V_{pk}}(\mathsf{gk}, \mathsf{pk}) = 1$ and the extractor outputs the correct td, then $\mathsf{Sim}(\mathsf{gk}, \mathsf{pk_e}, \mathsf{pk}, \theta, \mathsf{x})$ and $\mathsf{P}(\mathsf{gk}, \mathsf{pk_e}, \mathsf{pk}, \mathsf{x}; \mathsf{w})$ have the same distribution for any $\mathsf{x} = ([\boldsymbol{C}]_2, [\boldsymbol{C}']_2)$, $\mathsf{w} = (\sigma, \boldsymbol{t})$ in $\mathcal{R}_n^{sh}$.

We analyse each element of the proof independently.

1. For $i \in [1 \mathinner{..} n-1]$, $\hat{a}_i$ is chosen independently and uniformly at random in both distributions since $\hat{r}_i$ is picked uniformly at random. Moreover, in both distributions $\hat{a}_n = t_{sum} - \sum_{i=1}^{n-1} \hat{a}_i$ where t_{sum} equals $\sum_{i=1}^{n} \hat{P}_i$ in the honest case. Hence, $\hat{a}_n$ also has the same distribution.

2. Since Step 2 in V_{pk} is satisfied, then ϱ is non-zero. By similar reasoning as in the previous step, a_i is chosen independently and uniformly at random for $i \in [1 \mathinner{..} n]$ in both distributions.
3. Given that Step 3 and Step 11a are satisfied in V_{pk}, then $a_i = b_i$ for $i \in [1 \mathinner{..} n]$ in both distributions.
4. Given that Steps 6, 7, 8 9, 11b are satisfied, then the elements $[\beta^2 \varrho]_1$, $[\beta\hat{\beta}]_1$, and $[\beta^2 P_i + \beta\hat{\beta}\hat{P}_i]_1$, for $i \in [1 \mathinner{..} n]$, are well-formed (with respect to possibly malformed values P_i and $\hat{P}_i$). This is sufficient to show that $d_i = \beta^2 a_i + \beta\hat{\beta}\hat{a}_i$ for $i \in [1 \mathinner{..} n]$ in both distributions. Hence, d_i is uniquely determined by β, $\hat{\beta}$, a_i and $\hat{a}_i$.
5. Given that Steps 4, 10 and 11c are satisfied, then $[((P_i + P_0)^2 - 1)/\varrho]_1$ is well-formed (again, with respect to a possibly malformed P_i and P_0). Given this, we can verify that $e_i = ((a_i + P_0)^2 - 1)/\varrho$ in both distributions.
6. In both distributions, s is chosen independently and uniformly at random since $\hat{r}$ is picked uniformly at random.
7. Step 5 in V_{pk} guarantees that $\hat{P}_i = \theta^{2i}$ for $i \in [1 \mathinner{..} n]$. In that case, an honestly generated proof will always satisfy the verification equation on Step 3 in Fig. 3. Given that $\hat{\boldsymbol{a}}$, s and pk are fixed, then there is a unique value of $\boldsymbol{N}$ which satisfies that equation, and the simulator picks that exact value $\boldsymbol{N}$.

Hence the simulator's output and the prover's output have the same distribution. Thus tFLSZ is nn-ZK. $\square$

7 Implementation

We have created a reference implementation[6] to validate the protocol. The implementation uses Python 3.5+ and covers: (i) the computation of the public key (K in Fig. 3) together with the distributed key generation protocol (Fig. 2), (ii) the key verification algorithm V_{pk} (Fig. 10), and (iii) proof generation and verification (Fig. 3), along with the accompanying new unit vector argument (Fig. 5) and the permutation argument (Fig. 6). It follows our exposition closely, except for some of the local computations in the DKG protocol.

In particular, the complexity of computing polynomials $[\ell_i(\chi)]_k$ (and other related elements) from $\left[\chi^i\right]_k$ can be reduced from $\Theta(n^2)$ to $\Theta(n \log n)$ scalar multiplications using recursive procedures borrowed from FFT(see Appendix A for details). This however imposes the extra conditions that $(n+1) \mid (p-1)$ and $n+1$ is a power of 2. The current implementation uses a BN-256 curve[7], where the only value of $n > 1$ such that the conditions hold is $n = 3$. Work is in progress for moving to a different curve where $p-1$ is divisible by a large power of two. Note, nevertheless, that the correctness of the implementation, protocol testing, and verification of proofs is independent of this, as the output of local computations are not affected, only their efficiency.

[6] The code is open source and available at `https://github.com/grnet/lta_shuffle`
[7] As provided by OpenPairing, `https://github.com/dfaranha/OpenPairing`.

The multi-party computation of the public key is performed among k peers (bulletin board members) communicating via sockets (peers run the application from different terminals). Roughly speaking, each peer computes and shares their own part of the key with the rest, the final public key being the output of the distributed procedure explained in Section 3.2. For simulation purposes, the initial values for each peer, as well as their respective listening sockets, are derived from a configuration file. The total number of exchanged messages is independent of the number voters n and is equal to $9k(k-1)$.

Acknowledgements. This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 653497 (project PANORAMIX). Janno Siim was additionally supported by the Estonian Research Council grant PRG49. Part of this work was done while Prastudy Fauzi was working at Aarhus University and was supported by: the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC); the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583 (SODA).

References

ABL$^+$19a. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. DL-extractable UC-commitment schemes. In *ACNS 19*, LNCS, pages 385–405. Springer, Heidelberg, 2019.

ABL$^+$19b. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-secure CRS generation for SNARKs. In *AFRICACRYPT 19*, LNCS, pages 99–117. Springer, Heidelberg, 2019.

ABLZ17. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017.

ALSZ18. Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michał Zając. On qa-nizk in the bpk model. Cryptology ePrint Archive, Report 2018/877, 2018. `https://eprint.iacr.org/2018/877`.

BBH$^+$19. James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of kilian-based SNARGs. In *TCC 2019, Part II*, LNCS, pages 522–551. Springer, Heidelberg, March 2019.

BCG$^+$15. Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.

BD17. Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. Cryptology ePrint Archive, Report 2017/334, 2017. `http://eprint.iacr.org/2017/334`.

BDG$^+$13. Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. Why "Fiat-Shamir for proofs" lacks a proof. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 182–201. Springer, Heidelberg, March 2013.

BFS16. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016.

BG12. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, Heidelberg, April 2012.

BGG17. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017. `http://eprint.iacr.org/2017/602`.

CGGM00. Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000.

Cha81. David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

Dam92. Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, August 1992.

DGP$^+$19. Vanesa Daza, Alonso González, Zaira Pindado, Carla Ràfols, and Javier Silva. Shorter quadratic QA-NIZK proofs. In *PKC 2019, Part I*, LNCS, pages 314–343. Springer, Heidelberg, 2019.

FFHR19. Antonio Faonio, Dario Fiore, Javier Herranz, and Carla Ràfols. Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. LNCS, pages 159–190. Springer, Heidelberg, December 2019.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

FL16. Prastudy Fauzi and Helger Lipmaa. Efficient culpably sound NIZK shuffle argument without random oracles. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 200–216. Springer, Heidelberg, February / March 2016.

FLSZ17a. Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An efficient pairing-based shuffle argument. Cryptology ePrint Archive, Report 2017/894, 2017. `http://eprint.iacr.org/2017/894`.

FLSZ17b. Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An efficient pairing-based shuffle argument. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 97–127. Springer, Heidelberg, December 2017.

FLZ16. Prastudy Fauzi, Helger Lipmaa, and Michal Zajac. A shuffle argument secure in the generic model. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 841–872. Springer, Heidelberg, December 2016.

FS87. 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.

FS01. Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 368–387. Springer, Heidelberg, August 2001.

Fuc18. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018.

GJKR99. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999.

GK03. Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003.

GL07. Jens Groth and Steve Lu. A non-interactive shuffle with pairing based verifiability. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 51–67. Springer, Heidelberg, December 2007.

GO94. Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

GR16. Alonso González and Carla Ràfols. New techniques for non-interactive shuffle and range arguments. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 427–444. Springer, Heidelberg, June 2016.

Gro10. Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, October 2010.

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

LZ13. Helger Lipmaa and Bingsheng Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. *Journal of Computer Security*, 21(5):685–719, 2013.

MRV16. Paz Morillo, Carla Ràfols, and Jorge Luis Villar. The kernel matrix Diffie-Hellman assumption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 729–758. Springer, Heidelberg, December 2016.

Nao03. Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003.

TW10. Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 100–113. Springer, Heidelberg, May 2010.

Wee07. Hoeteck Wee. Lower bounds for non-interactive zero-knowledge. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 103–117. Springer, Heidelberg, February 2007.

A Computing Lagrange Polynomials

The circuit in Fig. 7 requires computing several Lagrange basis related elements: $\left[(\ell_i(\chi))_{i=1}^{n+1}\right]_z$ for $z \in \{1, 2\}$, $[(\beta\ell_i(\chi))_{i=1}^{n+1}]_1$, and also products of the basis

$[(\ell_i(\chi)^2)_{i=1}^{n+1}]_1$ and $[(\ell_i(\chi) \cdot \ell_{n+1}(\chi))_{i=1}^{n}]_1$. The naive way is to compute each of the elements as a linear combination of $\left[1, \chi, \chi^2, \dots\right]_z$ (or $[\beta, \beta\chi, \beta\chi^2, \dots]_1$). However, this would require $\Theta(n)$ scalar multiplications for each individual element and thus $\Theta(n^2)$ scalar multiplications altogether. We recall here some standard FFT-based tricks to improve the efficiency to $\Theta(n \log n)$ scalar multiplications. For this we need to set the points $\omega_1, \dots, \omega_{n+1}$ in the definition of Lagrange polynomials as $(n+1)$-th roots of unity.

A.1 Finding a Primitive Root of Unity

In order to compute the Lagrange basis $[\ell_i(\chi)]_1$ for $i \in [1 \mathinner{..} n+1]$ efficiently, we need to find the $(n+1)$-th primitive root of unity modulo p. This exists if and only if $(n+1) \mid (p-1)$. SNARK-friendly elliptic curves such as BLS12-381 have $p-1$ divisible by a large power of 2. Hence we can restrict $n+1$ to a power of 2. For example, if we expect 1 million voters, then we can take $n+1 = 2^{20} = 1,048,576$. Excessive positions in the shuffle can be filled with ciphertexts encrypting some special symbol which can be detected and removed during the decryption phase.

Given the above restriction, the $(n+1)$-th primitive root of unity modulo p can be computed as $\omega := g^{(p-1)/(n+1)}$ where g is a generator (primitive element) of the multiplicative group $\mathbb{Z}_p^*$. The generator itself can be found by sampling $g \leftarrow_{\$} \mathbb{Z}_p^*$ and testing that all prime factors q of $p-1$ satisfy $g^{(p-1)/q} \not\equiv 1 \pmod{p}$. This can be repeated until the property holds which will eventually happen since the multiplicative group of any finite field has a generator. Now we can define $\omega_i := \omega^i$ for $i \in [1 \mathinner{..} n+1]$.

A.2 Multi-point Evaluation

Our strategy for computing Lagrange basis is similar to Bowe et al. [BGG17]. We define four polynomials $f_k(Y) = \sum_{j=0}^{n} a_{k,j} Y^j$ for $k \in [1 \mathinner{..} 4]$ such that $[f_k(\omega^i)]_z$ are the elements we would like to compute and then apply an efficient FFT-based multi-point evaluation algorithm to each polynomial. However, we only know coefficients of the polynomials as group elements, but this turns out to be enough.

Let us first recall the following characterization of a Lagrange basis.

Theorem 7 ([BGG17], Claim 3.1). $\forall i \in [1 \mathinner{..} n+1]$, $\ell_i(X) = \sum_{j=0}^{n} \frac{\omega^{-ij}}{n+1} X^j$.

Let us define polynomials $f_1(Y) := \sum_{j=0}^{n} \frac{\chi^j}{n+1} Y^j$ and $f_2(Y) := \sum_{j=0}^{n} \frac{\beta\chi^j}{n+1} Y^j$. We may observe that based on the theorem above, $f_1(\omega^{-i}) = \ell_i(\chi)$ and $f_2(\omega^{-i}) = \beta\ell_i(\chi)$ for $i \in [1 \mathinner{..} n+1]$. Moreover, by just knowing $\left[\chi^j\right]_z$ (either $z = 1$ or $z = 2$) and $[\beta\chi^j]_1$, we can compute coefficients of f_1 and f_2 as group elements, i.e., $[a_{1,j}]_z := \left[\chi^j\right]_z/(n+1)$ and $[a_{2,j}]_1 := [\beta\chi^j]_1/(n+1)$. Slightly more work is needed to find similar polynomials for $\ell_i(\chi)^2$ and $\ell_i(\chi)\cdot\ell_{n+1}(\chi)$.

Since $\omega^{ij} = \omega^{i(n+1+j)}$ for $j \in [0\,..\,n-1]$, we can derive that

$$\ell_i(X)^2 = \frac{1}{(n+1)^2}\sum_{s=0}^{n}\sum_{t=0}^{n}\frac{1}{\omega^{i(s+t)}}X^{s+t} = \frac{1}{(n+1)^2}\sum_{j=0}^{2n}\frac{j+1}{\omega^{ij}}X^j$$

$$= \frac{X^n}{(n+1)\omega^{in}} + \frac{1}{(n+1)^2}\sum_{j=0}^{n-1}\frac{(j+1)X^j + (j+n+2)X^{j+n+1}}{\omega^{ij}}\ .$$

This allows us to define a polynomial $f_3(Y) := \frac{\chi^n}{n+1}Y^n + \frac{1}{(n+1)^2}\sum_{j=0}^{n-1}((j+1)\chi^j + (j+n+2)\chi^{j+n+1})Y^j$ such that $f_3(\omega^{-i}) = \ell_i^2(\chi)$ for $i \in [1\,..\,n+1]$. Again, we can compute coefficients as group elements: $[a_{3,n}]_1 = \frac{1}{n+1}[\chi^n]_1$ and $[a_{3,j}]_1 = \frac{j+1}{(n+1)^2}[\chi^j]_1 + \frac{j+n+2}{(n+1)^2}[\chi^{j+n+1}]_1$ for $j \in [0\,..\,n-1]$.

Finally, considering that $\omega^{n+1} = 1$, we can express

$$\ell_i(X)\ell_{n+1}(X) = \frac{1}{(n+1)^2}\sum_{s=0}^{n}\sum_{t=0}^{n}\frac{1}{\omega^{is}}\frac{1}{\omega^{(n+1)t}}X^{s+t} = \sum_{s=0}^{n}a_{4,s}(X)\frac{1}{\omega^{is}},$$

where $a_{4,s}(X) = \frac{1}{(n+1)^2}\sum_{t=0}^{n}X^{s+t}$ and define $f_4(Y) := \sum_{j=0}^{n}a_{4,j}Y^j$ with $a_{4,j} = a_{4,j}(\chi)$. Then $f_4(\omega^{-i}) = \ell_i(\chi)\ell_{n+1}(\chi)$ for $i \in [1\,..\,n+1]$. We can compute $[a_{4,j}]_1$ for $j \in [0\,..\,n]$ with the following recursive procedure which takes just $\theta(n)$ additions. First, we compute $[a_{4,0}]_1 = \frac{1}{(n+1)^2}\sum_{t=0}^{n}[\chi^t]_1$ and then we may observe that $[a_{4,j+1}]_1 = [a_{4,j}]_1 + [\chi^{j+n+1}]_1 - [\chi^j]_1$ which allows us to recursively compute all the rest of the values.

Now we are equipped to evaluate all of the Lagrange basis related elements in the public key by using a standard FFT-based multi-point evaluation algorithm in Fig. 11 which due to the master theorem for divide-and-conquer recurrences takes only $\Theta(n\log n)$ scalar multiplications. The only deviation from the text-book version is that coefficients of the polynomials are group elements and variables of the polynomials are field elements.

Now, considering that $\omega^{-i} = \omega^{n+1-i}$, we can compute

$$[(\ell_{n+1-j}(\chi))_{j=0}^{n}]_1 \leftarrow \mathsf{PolyEval}(\mathsf{gk}, \left([a_{1,j}]_1\right)_{j=0}^{n}, \omega, n),$$

$$[(\ell_{n+1-j}(\chi))_{j=0}^{n}]_2 \leftarrow \mathsf{PolyEval}(\mathsf{gk}, \left([a_{1,j}]_2\right)_{j=0}^{n}, \omega, n),$$

$$[\beta(\ell_{n+1-j}(\chi))_{j=0}^{n}]_1 \leftarrow \mathsf{PolyEval}(\mathsf{gk}, \left([a_{2,j}]_1\right)_{j=0}^{n}, \omega, n),$$

$$[(\ell_{n+1-j}^2(\chi))_{j=0}^{n}]_1 \leftarrow \mathsf{PolyEval}(\mathsf{gk}, \left([a_{3,j}]_1\right)_{j=0}^{n}, \omega, n),$$

$$[(\ell_{n+1-j}(\chi)\cdot\ell_{n+1}(\chi))_{j=0}^{n}]_1 \leftarrow \mathsf{PolyEval}(\mathsf{gk}, \left([a_{4,j}]_1\right)_{j=0}^{n}, \omega, n).$$

Therefore, all of the those elements can be computed with just $\Theta(n\log n)$ scalar multiplications.

```
$\mathsf{PolyEval}(\mathsf{gk}, [(a_i)_{i=0}^{n}]_z, \omega, n)$: $//n+1$ is a power of 2
  1. If $n = 1$: Return $\left([a_0]_z + \omega [a_1]_z, [a_0]_z + \omega^2 [a_1]_z\right)$
  2. $n' \leftarrow (n+1)/2$
  3. $[\boldsymbol{u}]_z \leftarrow \mathsf{PolyEval}(\mathsf{gk}, [(a_{2i})_{i=0}^{n'-1}]_z, \omega^2, n'-1)$
  4. $[\boldsymbol{v}]_z \leftarrow \mathsf{PolyEval}(\mathsf{gk}, [(a_{2i+1})_{i=0}^{n'-1}]_z, \omega^2, n'-1)$
  5. For $i \in [1 \mathinner{..} n']$: $[w_i]_z \leftarrow [u_i]_z + \omega^i [v_i]_z$
  6. For $i \in [n'+1 \mathinner{..} n+1]$: $[w_i]_z \leftarrow [u_{i-n'}]_z + \omega^i [v_{i-n'}]_z$
  7. Return $[\boldsymbol{w}]_z$
```

Fig. 11. Efficient multi-point evaluation of $[f(X)]_z = [\sum_{j=0}^{n} a_i X^j]_z$ at points $X = \omega^i$ for $i \in [1 \mathinner{..} n+1]$.

```
Commit Phase: Each party $\mathcal{P}_i$:
    1. Samples $\mathsf{td}_i = (\chi_i, \theta_i, \beta_i, \hat{\beta}_i, \varrho_i) \leftarrow \mathsf{K}_{\mathsf{td}}(\mathsf{gk})$ with $\mathsf{K}_{\mathsf{td}}$ from Fig. 3.
    2. Submits $\boldsymbol{c}_i \leftarrow \mathsf{dlCom}_{\mathsf{ck}}(\mathsf{td}_i)$ to BB.
Open Phase: Each party $\mathcal{P}_i$:
    1. Verifies that $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\mathrm{N}_p}$ on BB.
    2. Submits commitment opening together with $([\mathsf{td}_i]_1, [\mathsf{td}_i]_2)$ to BB.
Key Generation (Phase 1): For $i = 1$ to $\mathrm{N}_p$:
    1. If $i = 1$: $\mathcal{P}_1$ sets $\mathsf{pk}_1^{(0)} \leftarrow \mathsf{Setup}_1(\mathsf{gk})$.
    2. $\mathcal{P}_i$ submits $\mathsf{pk}_1^{(i)} \leftarrow \mathsf{Update}_1(\mathsf{gk}, \mathsf{pk}_1^{(i-1)}, \mathsf{td}_i)$ to BB.
Key Generation (Phase 2): For $i = 1$ to $\mathrm{N}_p$:
    1. If $i = 1$: $\mathcal{P}_1$ sets $\mathsf{pk}_2^{(0)} \leftarrow \mathsf{Setup}_2(\mathsf{pk}_1^{(\mathrm{N}_p)})$.
    2. $\mathcal{P}_i$ submits $\mathsf{pk}_2^{(i)} \leftarrow \mathsf{Update}_2(\mathsf{gk}, \mathsf{pk}_2^{(i-1)}, (\beta_i, \varrho_i))$ to BB.
Verification: By anyone:
    1. Set $\mathsf{pk}_1^{(0)} \leftarrow \mathsf{Setup}_1(\mathsf{gk})$ and $\mathsf{pk}_2^{(0)} \leftarrow \mathsf{Setup}_2(\mathsf{gk}, \mathsf{pk}_1^{(\mathrm{N}_p)})$.
    2. Verify that commitments are opened correctly.
    3. Check that for $i \in [1 \mathinner{..} \mathrm{N}_p]$, $\iota \in \mathsf{td}_i$: $[\iota]_1 \bullet [1]_2 = [1]_1 \bullet [\iota]_2$ and $[\iota]_1 \neq [0]_1$.
    4. For $i = 1$ to $\mathrm{N}_p$:
        (a) Check that $\mathsf{UpdateVer}_1(\mathsf{gk}, \mathsf{pk}_1^{(i-1)}, \mathsf{pk}_1^{i}, ([\mathsf{td}_i]_1, [\mathsf{td}_i]_2)) = 1$.
        (b) Check that $\mathsf{UpdateVer}_2(\mathsf{gk}, \mathsf{pk}_2^{(i-1)}, \mathsf{pk}_2^{i}, [\varrho_i, \beta_i]_2) = 1$.
    5. Return $\mathsf{Finalize}(\mathsf{pk}_1^{(\mathrm{N}_p)}, \mathsf{pk}_2^{(\mathrm{N}_p)})$
```

Fig. 12. DKG Protocol for tFLSZ

B DKG Protocol for tFLSZ

For concreteness, we provide the complete DKG protocol for tFLSZ in Fig. 12. The protocol here is an instantiation of the general protocol by Abdolmaleki et al. [ABL$^+$19b] which UC-securely realises the CRS functionality for any verification-friendly CRS. We assume to have access to a secure append-only storage system that we call a *bulletin board (BB)* (more generally, any secure broadcast protocol suits here, as was suggested by Abdolmaleki et al.) and to any DL-extractable commitment $\mathsf{dlCom}_{\mathsf{ck}}$. We note that the DL-extractable commitment of [ABL$^+$19a] is an interactive protocol and therefore each pair of parties should run it. For simplicity we represent it in the figure as a non-interactive commitment.

The protocol starts with each party picking trapdoor shares and submitting a commitment of it to the bulletin board. Once all parties have submitted the commitment starts the open phase where parties open their commitments to group elements (since this is DL-extractable commitment). The point of these first two phases is to guarantee that shares are picked independently. This is followed by two phases of public key generation described respectively in Fig. 13 and Fig. 14. Each of the phases takes N_p rounds where N_p is the number of parties in the DKG protocol. A key generation phase $k \in \{1, 2\}$ contains:

$\mathsf{Setup}_k(\mathsf{gk})$: Publicly computable function which for $k = 1$ initiates the public key with all 1s and for $k = 2$ evaluates those linear combination gates of layer L_1 (as described in Fig. 7) that are required by the subsequent layer C_2 as an input.

$\mathsf{Update}_k(\mathsf{gk}, \mathsf{pk}_k^{(i-1)}, \mathsf{td}_i)$: On an input the public key $\mathsf{pk}_k^{(i)}$ of the previous party and trapdoor shares td_i of the current party, updates elements corresponding to multdiv layer C_k with shares td_i by multiplying.

$\mathsf{UpdateVer}_k(\mathsf{pk}_k^{(i-1)}, \mathsf{pk}_k^{(i)}, ([\mathsf{td}_i]_1, [\mathsf{td}_i]_2))$: Verifies that the $\mathcal{P}_i$ computed Update_k correctly with respect to his shares td_i.

$\mathsf{Finalize}(\mathsf{pk}_1^{(\mathrm{N}_p)}, \mathsf{pk}_2^{(\mathrm{N}_p)})$: Takes as an input the last output $\mathsf{pk}_1^{(\mathrm{N}_p)}$ of phase 1 and the last output $\mathsf{pk}_2^{(\mathrm{N}_p)}$ of phase 2 (i.e., the outputs of party $\mathcal{P}_{\mathrm{N}_p}$), computes the remaining linear combination gates of layer L_1, and composes everything into a well-formed public key pk.

Finally, each party in the protocol runs the verification phase which outputs the key and verifies that all parties followed the protocol.

C Security Proof of Unit Vector Argument

Theorem 8. *The unit vector argument (see Fig. 5) in the CRS model has perfect completeness and perfect zero-knowledge with respect to the simulator in Fig. 15.*

Proof. Perfect completeness. This is easy to verify by plugging honestly generated values to the verification equations. From the right hand side of the equation in step 2 of Fig. 5 we can derive $[a, \hat{a}]_1 \bullet [\beta^2, \beta\hat{\beta}]_2^\top = [\beta^2 P_I + \beta^2 r\varrho + \beta\hat{\beta}\hat{P}_I + \beta\hat{\beta}\hat{r}]_T = [d]_1 \bullet [1]_2$. On the left hand side of the equation in step 3 of Fig. 5, considering

```
$\mathsf{Setup}_1(\mathsf{gk})$: Return $\mathsf{pk}_1 \leftarrow \big([1,\dots,1]_1, [1,\dots,1]_2\big) \in \mathbb{G}_1^{6n+6} \times \mathbb{G}_2^{n+6}$.
$\mathsf{Update}_1(\mathsf{gk}, \mathsf{pk}_1, (\chi_u, \theta_u, \beta_u, \hat{\beta}_u, \varrho_u))$:
    1. Parse $\mathsf{pk}_1 = \begin{pmatrix} [\beta, \hat{\beta}, \varrho, \beta\varrho, \beta^2\varrho, \beta\hat{\beta}, (\chi^i, \theta^i)_{i=1}^{2n}, (\beta\chi^i, \gamma\theta^{2i})_{i=1}^{n}]_1 \\ [\theta, \beta, \hat{\beta}, \varrho, \beta^2, \beta\hat{\beta}, (\chi^i)_{i=1}^{n}]_2 \end{pmatrix}$.
    2. Return
       $\widetilde{\mathsf{pk}}_1 \leftarrow \begin{pmatrix} \beta_u[\beta]_1, \hat{\beta}_u[\hat{\beta}]_1, \varrho_u[\varrho]_1, (\beta_u\varrho_u)[\beta\varrho]_1, ((\beta_u^2\varrho_u)[\beta^2\varrho]_1, (\beta_u\hat{\beta}_u)[\beta\hat{\beta}]_1, \\ \big(\chi_u^i[\chi^i]_1, \theta_u^i[\theta^i]_1\big)_{i=1}^{2n}, \big((\beta_u\chi_u^i)[\beta\chi^i]_1, (\hat{\beta}_u\theta_u^{2i})[\hat{\beta}\theta^{2i}]_1\big)_{i=1}^{n} \\ \theta_u[\theta]_2, \beta_u[\beta]_2, \hat{\beta}_u[\hat{\hat{\beta}}]_2, \varrho_u[\varrho]_2, \beta_u^2[\beta^2]_2, \beta_u\hat{\beta}_u[\beta\hat{\beta}]_2, (\chi_u^i[\chi^i]_2)_{i=1}^{n} \end{pmatrix}$.
$\mathsf{UpdateVer}_1(\mathsf{gk}, \mathsf{pk}_1, \widetilde{\mathsf{pk}}_1, \pi_{\widetilde{\mathsf{pk}}_1})$:
    1. Parse $\mathsf{pk}_1$, $\widetilde{\mathsf{pk}}_1$ as above.
    2. Parse $\pi_{\widetilde{\mathsf{pk}}_1}$ as $([\chi_u, \theta_u, \beta_u, \hat{\beta}_u, \varrho_u]_1, [\chi_u, \theta_u, \beta_u, \hat{\beta}_u, \varrho_u]_2)$.
    3. For $\iota \in \{\chi, \theta, \beta, \hat{\beta}, \varrho\}$: check
        (a) $[\iota\iota_u]_1 \bullet [1]_2 = [\iota]_1 \bullet [\iota_u]_2$,
        (b) $[1]_1 \bullet [\iota\tilde{\iota}]_2 = [\iota\tilde{\iota}]_1 \bullet [1]_2$.
    4. Check $[\beta_u\varrho_u\beta\varrho]_1 \bullet [1]_2 = [\beta_u\beta]_1 \bullet [\varrho_u\varrho]_2$.
    5. Check $[\beta_u^2\varrho_u\beta^2\varrho]_1 \bullet [1]_2 = [\beta_u\varrho_u\beta\varrho]_1 \bullet [\beta_u\beta]_2$.
    6. Check $[\beta_u\hat{\beta}_u\beta\hat{\beta}]_1 \bullet [1]_2 = [\beta_u\hat{\beta}_u]_1 \bullet [\beta\hat{\beta}]_2$.
    7. Check $[1]_1 \bullet [\beta_u\hat{\beta}_u\beta\hat{\beta}]_2 = [\beta_u\hat{\beta}_u]_1 \bullet [\beta\hat{\beta}]_2$.
    8. Check $[1]_1 \bullet [\beta_u^2\beta^2]_2 = [\beta_u\beta]_1 \bullet [\beta_u\beta]_2$.
    9. For $i = 2$ to $2n$: check
        (a) $[\chi_u^i\chi^i]_1 \bullet [1]_1 = [\chi_u^{i-1}\chi^{i-1}]_1 \bullet [\chi_u\chi]_2$,
        (b) $[\theta_u^i\theta^i]_1 \bullet [1]_1 = [\theta_u^{i-1}\theta^{i-1}]_1 \bullet [\theta_u\theta]_2$,
    10. For $i = 2$ to $n$: check
        (a) $[\beta_u\chi_u^i\beta\chi^i]_1 \bullet [1]_1 = [\chi_u^i\chi^i]_1 \bullet [\beta_u\beta]_2$,
        (b) $[\hat{\beta}_u\theta_u^{2i}\hat{\beta}\theta^{2i}]_1 \bullet [1]_1 = [\theta_u^{2i}\theta^{2i}]_1 \bullet [\hat{\beta}_u\hat{\beta}]_2$,
        (c) $[1]_1 \bullet [\chi_u^i\chi_u^i]_2 = [\chi_u^i\chi^i]_1 \bullet [1]_2$.
```

Fig. 13. Phase 1 algorithms for DKG

that $a = b$, we get

$$\begin{aligned}
[(a + P_0)^2 - \alpha^2]_T &= [(P_I + r\varrho + P_0)^2 - \alpha^2]_T \\
&= [(P_I + P_0)^2 + 2r(P_I + P_0)\varrho + (r\varrho)^2 - \alpha^2]_T \\
&= [(P_I + P_0)^2 + r(2(P_I + P_0)\varrho + r\varrho^2) - \alpha^2 + (1 - 1)]_T \\
&= [r(2(P_I + P_0)\varrho + r\varrho^2) + ((P_I + P_0)^2 - 1) + (1 - \alpha^2)]_T \\
&= \varrho[r(2(P_I + P_0) + r\varrho) + ((P_I + P_0)^2 - 1)/\varrho]_T + [(1 - \alpha^2)]_T \\
&= \varrho[r(2(a + P_0) - r\varrho) + ((P_I + P_0)^2 - 1)/\varrho]_T + [(1 - \alpha^2)]_T \\
&= [e]_1 \bullet [\varrho]_2 + (1 - \alpha^2)[1]_T .
\end{aligned}$$

Perfect zero-knowledge. Let $([\hat{a}]_1, (I, \hat{r})) \in \mathcal{R}_{uv}$ and $(\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{K}_{uv}(\mathsf{gk}, n)$. We show that $\mathsf{P}_{uv}(\mathsf{gk}, \mathsf{pk}, [\hat{a}]_1, (I, \hat{r}))$ and $\mathsf{Sim}_{uv}(\mathsf{gk}, (\beta, \hat{\beta}), [\hat{a}]_1)$ are identical distributions. In both cases $[a]_1$ is a uniformly random group element and $[b]_2 = [a]_2$. Given that the latter are fixed, there is exactly one possible value of $[e]_1$

```
$\mathsf{Setup}_2(\mathsf{gk}, \mathsf{pk}_1)$:
    1. Parse $\mathsf{pk}_1$ as in Step 1 of $\mathsf{Update}_1$.
    2. Compute $[\ell_i(\chi)]_1$, $[\beta\ell_i(\chi)]_1$, $[\ell_i(\chi)^2]_1$, $[\ell_i(\chi) \cdot \ell_{n+1}(\chi)]_1$ for $i \in [1 \mathinner{..} n+1]$ as a linear combination of $[(\chi^i)_{i=0}^{2n}]_1$ and $[(\beta\chi^i)_{i=0}^{n}]_1$. (see Appendix A).
    3. For $i = 1$ to $n$:
        (a) $[Q_i]_1 \leftarrow 4[\ell_i(\chi)^2]_1 + 4[\ell_{n+1}(\chi)^2]_1 + 8[\ell_i(\chi) \cdot \ell_{n+1}(\chi)]_1 - 4[\ell_i(\chi)]_1 - 4[\ell_{n+1}(\chi)]_1$.
        (b) $[\beta P_i + \hat{\beta}\hat{P}_i]_1 \leftarrow 2[\beta\ell_i(\chi)]_1 + [\beta\ell_{n+1}(\chi)]_1 + [\hat{\beta}\theta^{2i}]_1$
    4. Return $\mathsf{pk}_2 \leftarrow [\boldsymbol{Q}, \beta\boldsymbol{P} + \hat{\beta}\hat{\boldsymbol{P}}]_1$.
$\mathsf{Update}_2(\mathsf{gk}, \mathsf{pk}_2, (\beta_u, \varrho_u))$:
    1. Parse $\mathsf{pk}_2 = [\boldsymbol{Q}/\varrho', \ \beta'(\beta\boldsymbol{P} + \hat{\beta}\hat{\boldsymbol{P}})]_1$.
    2. Return $\widetilde{\mathsf{pk}}_2 \leftarrow \big((1/\varrho_u)[\boldsymbol{Q}/\varrho']_1, \ \beta_u[\beta'(\beta\boldsymbol{P} + \hat{\beta}\hat{\boldsymbol{P}}]_1\big)$.
$\mathsf{UpdateVer}_2(\mathsf{gk}, \mathsf{pk}_2, \widetilde{\mathsf{pk}}_2, \pi_{\widetilde{\mathsf{pk}}_2})$:
    1. Parse $\mathsf{pk}_2$, $\widetilde{\mathsf{pk}}_2$ as above and $\pi_{\widetilde{\mathsf{pk}}_2}$ as $[\varrho_u, \beta_u]_2$.
    2. Check $[\beta_u\beta'(\beta\boldsymbol{P} + \hat{\beta}\hat{P})]_1 \bullet [1]_2 = [\beta'(\beta\boldsymbol{P} + \hat{\beta}\hat{P})]_1 \bullet [\beta_u]_2$.
    3. Check $[\boldsymbol{Q}/(\varrho'\varrho_u)]_1 \bullet [\varrho_u]_2 = [\boldsymbol{Q}/\varrho']_1 \bullet [1]_2$.
$\mathsf{Finalize}(\mathsf{pk}_1, \mathsf{pk}_2)$:
    1. Parse $\mathsf{pk}_1$ as in $\mathsf{Update}_1$ and $\mathsf{pk}_2$ as $[\boldsymbol{Q}/\varrho, \beta^2\boldsymbol{P} + \beta\hat{\beta}\hat{\boldsymbol{P}}]_1$.
    2. Compute $[\ell_i(\chi)]_1$, $[\ell_i(\chi)]_2$ for $i \in [1 \mathinner{..} n+1]$ as a linear combination of $[(\chi^i)_{i=0}^{n}]_1$ and $[(\chi^i)_{i=0}^{n}]_2$. (see Appendix A).
    3. $[P_0]_1 \leftarrow [\ell_{n+1}(\chi)]_1 - [1]_1$ , $[P_0]_2 \leftarrow [\ell_{n+1}(\chi)]_2 - [1]_2$.
    4. For $i = 1$ to $n$: $[P_i]_1 \leftarrow 2[\ell_i(\chi)]_1 + [\ell_{n+1}(\chi)]_1$, $[P_i]_2 \leftarrow 2[\ell_i(\chi)]_2 + [\ell_{n+1}(\chi)]_2$
    5. $\mathsf{pk}_{uv} \leftarrow \begin{pmatrix} [1, \ P_0, \ \boldsymbol{P}, \ \varrho, \ \boldsymbol{Q}/\varrho, \ \sum_{i=1}^{n} \hat{P}_i, \ \beta^2\varrho, \ \beta\hat{\beta}, \ \beta^2\boldsymbol{P} + \beta\hat{\beta}\hat{\boldsymbol{P}}]_1, \\ [1, \ P_0, \ \boldsymbol{P}, \ \varrho, \ \beta^2, \ \beta\hat{\beta}]_2, \ [1]_1 \bullet [1]_2 \end{pmatrix}$.
    6. $\mathsf{pk}_{pkv} \leftarrow \big([(\theta^{2i-1})_{i=1}^{n}, \ (\ell_i(\chi))_{i=1}^{n+1}, \ \beta, \ \hat{\beta}, \ (\chi^i)_{i=1}^{n+1}]_1, \ [\chi, \theta, \beta, \hat{\beta}]_2\big)$.
    7. Return $([\hat{\boldsymbol{P}}]_1, \mathsf{pk}_{uv}, \mathsf{pk}_{pkv})$. //Note that $\mathsf{pk}_{vf}$ is never used
```

Fig. 14. Phase 2 algorithms for DKG

```
$\mathsf{Sim}_{uv}(\mathsf{gk}, \mathsf{pk}, (\beta, \hat{\beta}), [\hat{a}]_1)$
    1. $r \leftarrow_{\$} \mathbb{F}_p$, $[a]_1 \leftarrow [P_1]_1 + r[\varrho]_1$, $[b]_2 \leftarrow [P_1]_2 + r[\varrho]_2$.
    2. $[e]_1 \leftarrow r \cdot (2([a]_1 + [P_0]_1) - r[\varrho]_1) + [\big((P_1 + P_0)^2 - 1\big)/\varrho]_1$.
    3. $[d]_1 \leftarrow \beta^2[a]_1 + \beta\hat{\beta}[\hat{a}]_1$.
    4. Return $([d]_1, [a]_1, [b]_2, [e]_1)$.
```

Fig. 15. Simulator of unit vector argument

that would satisfy the verification equation on Step 3 of V_{uv} and Sim_{uv} picks that value. Finally, since honest $[d]_1 = [\beta^2 P_I + \beta\hat{\beta}\hat{P}_I]_1 + r[\beta^2\varrho]_1 + \hat{r}[\beta\hat{\beta}]_1 = [\beta^2(P_I + r\varrho) + \beta\hat{\beta}(\hat{P}_I + \hat{r})]_1$, then $[a, \hat{a}]_1$ and $(\beta, \hat{\beta})$ uniquely determine $[d]_1$, namely, Sim_{uv} picks it as $[d]_1 = \beta^2[a]_1 + \beta\hat{\beta}[\hat{a}]_1$. □

In the rest of this section, we follow the ideas from [FKL18, Section 7] and prove the following result.

Theorem 9. *If the $(3n-1)$-PDL assumption holds, then the unit vector argument has computational knowledge-soundness in the AGM.*

We define a notion of *idealised verification*, where the verification equation holds true for polynomials (with trapdoor elements as variables) rather than for polynomial evaluations only (*real verification*, for concrete trapdoor elements). We show that no element outside the language (i.e., no non-unit vectors) can pass the idealised verification and if an adversary manages to pass the real verification but no ideal one, then she can be used to break the $(3n-1)$-PDL assumption.

C.1 Idealised Verification

In the idealised verification the unit vector argument verification equations described in Step 3 of Fig. 5 hold for polynomials. That is, instead of concrete 5 group elements (an instance x and proof $\pi = ([a, d, e]_1, [b]_2))$, the adversary submits 5 polynomials $Z(\boldsymbol{X})$ (corresponding to the instance x), and $A(\boldsymbol{X})$, $B(\boldsymbol{X})$, $D(\boldsymbol{X})$, $E(\boldsymbol{X})$ (constituting to the proof elements), such that the verification equations hold as polynomials:

$$D(\boldsymbol{X}) - A(\boldsymbol{X})X_{\hat{\beta}}^2 - Z(\boldsymbol{X})X_\beta X_{\hat{\beta}}^2, \tag{1}$$

$$(A(\boldsymbol{X}) + P_0(X))(B(\boldsymbol{X}) + P_0(X)) - X_\varrho C(\boldsymbol{X}) - 1 + X_\alpha(B(\boldsymbol{X}) - A(\boldsymbol{X})) = 0 \ . \tag{2}$$

The adversary does not have full freedom in designing the polynomials but can create them as affine combinations of CRS elements. In [FKL18] such an adversary is called *affine*, and an explanation is given why considering such limited adversaries is actually enough. Thus, the polynomials $A(\boldsymbol{X}), D(\boldsymbol{X})$ and $E(\boldsymbol{X})$ are of form $G(\boldsymbol{X})$, cf. Eq. 3, and $B(\boldsymbol{X})$ of form $H(\boldsymbol{X})$, cf. Eq. 4, where $\boldsymbol{X} = (X, X_\theta, X_\beta, X_{\hat{\beta}}, X_\varrho, X_{\mathsf{sk_e}})$ denotes the formal variables corresponding to the trapdoor td.

$$\begin{aligned} G(\boldsymbol{X}) :=& \sum_{\iota \in S_1} G_\iota \cdot M(\boldsymbol{X}, \iota) + g_1(X) + \tilde{g}(X) + g_Q(X)/X_\varrho + g_\theta(X_\theta) + \\ &+ g_{sm}(\boldsymbol{X}) + X_\beta \cdot g_2(X) + X_{\hat{\beta}} \cdot \hat{g}(X_\theta) \end{aligned} \tag{3}$$

$$H(\boldsymbol{X}) := \sum_{\iota \in S_2} H_\iota \cdot M(\boldsymbol{X}, \iota) + h(X) \tag{4}$$

where

- $M(\boldsymbol{X}, \iota)$ is the monomial $X^{i_1} X_\theta^{i_2} X_\beta^{i_3} X_{\hat{\beta}}^{i_4} X_\varrho^{i_5} X_{\mathsf{sk_e}}^{i_6}$ for a symbolic value $\iota = \chi^{i_1}\theta^{i_2}\beta^{i_3}\hat{\beta}^{i_4}\varrho^{i_5}\mathsf{sk_e}^{i_6}$.
- $S_1 = \{\beta, \hat{\beta}, \varrho, \beta\varrho, \beta^2\varrho, \beta\hat{\beta}\}$ and $S_2 = \{\theta, \beta, \hat{\beta}, \varrho, \mathsf{sk_e}, \beta^2, \beta\hat{\beta}\}$.

– $G_\iota, H_\iota \in \mathbb{F}_p$, $g_1, g_2, h \in \mathsf{Span}(\{X^i\}_{i=0}^n)$, $\tilde{g} \in \mathsf{Span}(\{X^i\}_{i=n+1}^{2n})$, $g_Q \in \mathsf{Span}(\{(P_i(X) + P_0(X))^2 - 1\}_{i=1}^n)$, $\hat{g} \in \mathsf{Span}(\{\hat{P}_i(X_\theta)\}_{i=1}^n)$, $g_\theta \in \mathsf{Span}(\{X_\theta^i)\}_{i=1}^{2n})$, $g_{sm} \in \mathsf{Span}(\{X_\beta^2 P_i(X) + X_\beta X_{\hat{\beta}} \hat{P}_i(X_\theta)\}_{i=1}^n)$.

We show that if that is the case, then the unit vector argument holds.

Suppose that a PPT adversary $\mathcal{A}$ given gk outputs an instance polynomial $Z(\boldsymbol{X})$ and the corresponding proof polynomials $A(\boldsymbol{X}), B(\boldsymbol{X}), D(\boldsymbol{X}), E(\boldsymbol{X})$.

Since all random variables in $\boldsymbol{X}$ are independent and from verification equation 2 follows that

$$T_1(\boldsymbol{X}) := (A(\boldsymbol{X}) + P_0(X))(B(\boldsymbol{X}) + P_0(X)) - X_\varrho \cdot C(\boldsymbol{X}) - 1 = 0\,, \text{ and}$$
$$T_2(\boldsymbol{X}) := B(\boldsymbol{X}) - A(\boldsymbol{X}) = 0\,.$$

It follows that $A(\boldsymbol{X}) = B(\boldsymbol{X}) = \sum_{\iota \in \{\theta, \beta, \hat{\beta}, \varrho\}} A_\iota X_\iota + A_{\beta\hat{\beta}} X_\beta X_{\hat{\beta}} + a_1(X)$. Now $T_1(\boldsymbol{X}) = (A(\boldsymbol{X}) + P_0(X))^2 - X_\varrho \cdot C(\boldsymbol{X}) - 1 = 0$. We may observe that in $T_1(\boldsymbol{X})$ coefficients of monomials X_θ^2, X_β^2, $X_{\hat{\beta}}^2$, $X_\beta^2 X_{\hat{\beta}}^2$ are respectively $A_\delta^2 = 0$, $A_\beta^2 = 0$, $A_{\hat{\beta}}^2 = 0$, $A_{\beta\hat{\beta}}^2 = 0$. Therefore we can simplify $A(\boldsymbol{X}) = a_1(X) + A_\varrho X_\varrho = \sum_{i=0}^n A_i P_i(X) + A_\varrho X_\varrho$. Moreover, let us denote $\boldsymbol{X}' = (X_\theta, X_\beta, X_{\hat{\beta}}, X_\varrho, X_{\mathsf{sk_e}})$. Then the constant value in $T_1(\boldsymbol{X}')$ is

$$(\sum_{i=0}^n A_i P_i(X) + P_0(X))^2 - 1 - c_Q(X) = 0 \ . \tag{5}$$

Let us now turn to the Eq. 1 that guarantees $X_\beta^2 \cdot A(\boldsymbol{X}) + X_\beta X_{\hat{\beta}} \cdot Z(\boldsymbol{X}) - D(\boldsymbol{X}) = 0$. We multiply it by X_ϱ to remove the possible division in $Z(\boldsymbol{X})$ and $D(\boldsymbol{X})$, then

$$X_\beta^2 \cdot (X_\varrho A(\boldsymbol{X})) + X_\beta X_{\hat{\beta}} \cdot (X_\varrho Z(\boldsymbol{X})) - X_\varrho D(\boldsymbol{X}) = 0. \tag{6}$$

All the non-zero monomials of $X_\varrho D(\boldsymbol{X})$ must contain X_β^2 or $X_\beta X_{\hat{\beta}}$, hence $X_\varrho D(\boldsymbol{X}) = D_{\beta^2\varrho} X_\beta^2 X_\varrho^2 + D_{\beta\hat{\beta}} X_\beta X_{\hat{\beta}} X_\varrho + X_\varrho \cdot d_{sm}(\boldsymbol{X})$ where $d_{sm}(\boldsymbol{X}) = \sum_{i=1}^n D_i \cdot (X_\beta^2 P_i(X) + X_\beta X_{\hat{\beta}} \hat{P}_i(X_\theta))$. Additionally, let us consider that $X_\varrho X_\beta^2 \cdot A(\boldsymbol{X}) = \sum_{i=0}^n A_i P_i(X) X_\varrho X_\beta^2 + A_\varrho X_\varrho^2 X_\beta^2$. Hence $X_\varrho X_\beta X_{\hat{\beta}} Z(\boldsymbol{X})$ can only contain monomials present in $X_\varrho X_\beta^2 \cdot A(\boldsymbol{X})$ or $X_\varrho D(\boldsymbol{X})$ that are divisible by $X_\beta X_{\hat{\beta}}$ i.e., $Z(\boldsymbol{X}) = Z_0 X^0 + \sum_{i=1}^n Z_i \hat{P}_i(X_\theta)$. Then substituting into Eq. 6 we get that that

$$A_0 P_0(X) X_\varrho X_\beta^2 + \sum_{i=1}^n (A_i - D_i) P_i(X) X_\varrho X_\beta^2 + (A_\varrho - D_{\beta^2\varrho}) X_\beta^2 X_\varrho^2 +$$
$$(Z_0 - D_{\beta\hat{\beta}}) X_\beta X_{\hat{\beta}} X_\varrho + \sum_{i=1}^n (Z_i - D_i) \hat{P}_i(X_\theta) X_\beta X_{\hat{\beta}} X_\varrho = 0.$$

Since the above monomials are linearly independent, then $A_0 = 0$, $A_i = D_i = Z_i$ for $i \in [1 \,..\, n]$, $A_\varrho = D_{\beta^2\varrho}$, and $Z_0 = D_{\beta\hat{\beta}}$. Now $A(\boldsymbol{X}) = \sum_{i=1}^n a_i P_i(X) + A_\varrho X_\varrho$

and then from Eq. (5) we get also $(\sum_{i=i}^{n} a_i P_i(X) + P_0(X))^2 - 1 = c_Q(X)$ which together with Lemma 1 implies that $(A_1, \dots, A_n)$ is a unit vector. Considering that $A_i = Z_i$, we can extract I and Z_0 such that $[\hat{a}]_1 = [\hat{P}_I + Z_0]_1$.

C.2 Security Against Algebraic Adversaries

We show the security against algebraic adversaries by contradiction. That is, we assume an adversary $\mathcal{A}$ breaks the knowledge-soundness, i.e., she provides an instance and corresponding valid proof such that no extractor can efficiently extract the witness. However, as shown in Appendix C.1, if the verification equations hold as polynomials (cf. Eq. 1 and 1), then the extractor can always obtain the correct witness. Thus, it must be the case that verification equations as defined in Fig. 5 accept, but only as polynomial evaluations at trapdoor td, not as polynomials (at least one of two verification equation polynomials is non-zero). Using this distinction, we construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ to break the $(3n-1)$-PDL assumption.

First of all, we define algebraic adversaries:

Definition 10 (Algebraic adversary). *Let $[\boldsymbol{x_1}]_1, [\boldsymbol{x_2}]_2$ be vectors of group elements. We call an adversary $\mathcal{A}$* algebraic *if on input $[\boldsymbol{x_1}]_1, [\boldsymbol{x_2}]_2$ it returns $[\boldsymbol{y_1}]_1, [\boldsymbol{y_2}]_1, \boldsymbol{Z_1}, \boldsymbol{Z_2}$ such that $[\boldsymbol{y}_1]_1 = \boldsymbol{Z_1} \cdot [\boldsymbol{x_1}]_1$ and $[\boldsymbol{y}_2]_2 = \boldsymbol{Z_2} \cdot [\boldsymbol{x_2}]_1$ for some matrices $\boldsymbol{Z}_1, \boldsymbol{Z}_2$ over $\mathbb{F}_p$.*

Intuitively, we call an adversary $\mathcal{A}$ algebraic if for each output group element she also outputs a linear representation in the input group elements. An algebraic adversary is stronger than a generic group model adversary since $\mathcal{A}$ sees the bit-level structure of the group elements. However, this also means that proofs in the AGM are reductions to assumptions rather than unconditional proofs like in the generic group model.

Assume that an algebraic adversary $\mathcal{A}$ breaks the soundness of the unit vector argument. Then there exists an adversary $\mathcal{B}$ that breaks the $(3n-1)$-PDL assumption and proceeds as follows.

First, $\mathcal{B}$ gets as input two vectors of group elements $[(z^i)_{i=0}^{3n-1}]_1, [(z^i)_{i=0}^{3n-1}]_2$. Second, she sets

$$\chi = r_1 z + s_1 \,, \qquad \tau = r_2 z + s_2 \,, \qquad \beta = r_3 z + s_3 \,,$$
$$\hat{\beta} = r_4 z + s_4 \,, \qquad \varrho = r_5 z + s_5$$

for some randomly picked $\{r_i, s_i\}_{i=1}^{5}$, and builds a CRS for the unit vector argument that will be provided for $\mathcal{A}$. Since the adversary $\mathcal{A}$ is algebraic she returns elements $[\hat{a}]_1$, $[a]_1$, $[d]_1$, $[e]_1$, $[b]_2$ as her instance and proof along with vectors $\{\boldsymbol{z}_\iota\}_{\iota \in \hat{a}, a, b, d, e}$ – their representation as the combination of the CRS elements.

Let R_1 and R_2 be the verification equations. Since the adversary $\mathcal{B}$ picked s_i, r_i on her own, $\mathcal{B}$ knows all coefficients of R_1 and R_2 as polynomials in Z – a formal variable corresponding to z. The degree of R_1, R_2 is upper-bounded by the highest degree of the CRS element from $\mathbb{G}_1$, $2n-1$, plus the highest degree of the CRS element from $\mathbb{G}_2$, n. Thus $\deg(R_i)$ is at most $3n-1$. Let $R \in \{R_1, R_2\}$ be the equation that holds for the concrete z, i.e., $R(z) = 0$ but does not hold as a polynomial in Z, i.e., $R(Z) \neq 0$. Then $\mathcal{B}$ can factor R and find z', such that $R(z') = 0$. With probability at least $1/(3n-1))$ holds $z = z'$.

With this, we have proven the result in Theorem 9.