YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model

Ignacio Cascudo[1*], Bernardo David[2†], Lydia Garms[1 ‡], and Anders Konring[2 §]

[1] IMDEA Software Institute, Madrid, Spain.
{ignacio.cascudo, lydia.garms}@imdea.org
[2] IT University of Copenhagen, Copenhagen, Denmark.
bernardo@bmdavid.com, konr@itu.dk

Abstract. Achieving adaptive (or proactive) security in cryptographic protocols is notoriously difficult due to the adversary's power to dynamically corrupt parties as the execution progresses. Inspired by the work of Benhamouda *et al.* in TCC 2020, Gentry *et al.* in CRYPTO 2021 introduced the YOSO (You Only Speak Once) model for constructing adaptively (or proactively) secure protocols in massively distributed settings (*e.g.* blockchains). In this model, instead of having all parties execute an entire protocol, smaller *anonymous committees* are randomly chosen to execute each individual round of the protocol. After playing their role, parties encrypt protocol messages towards the the next anonymous committee and erase their internal state before publishing their ciphertexts. However, a big challenge remains in realizing YOSO protocols: *efficiently* encrypting messages towards anonymous parties selected at random without learning their identities, while proving the encrypted messages are valid w.r.t. the protocol. In particular, the protocols of Benhamouda *et al.* and of Gentry *et al.* require showing ciphertexts contain valid shares of secret states. We propose concretely efficient methods for encrypting a protocol's secret state towards a random anonymous committee. We start by proposing a very simple and efficient scheme for encrypting messages towards randomly and anonymously selected parties. We then show constructions of publicly verifiable secret (re-)sharing (PVSS) schemes with concretely efficient proofs of (re-)share validity that can be generically instantiated from encryption schemes with certain linear homomorphic properties. Finally, we show that our PVSS schemes can be efficiently realized from our encyption scheme.

1 Introduction

Cryptographic protocols traditionally rely on secure channels among parties whose identities are publicly known. However, while knowing parties' identities makes it easy to construct secure channels, it also makes it easy for an adaptive (or mobile) adversary to corrupt parties as a protocol execution proceeds. Recently, an elegant solution for this problem has been suggested [2,16]: instead of keeping secret state throughout the execution, parties periodically transfer their state to randomly selected anonymous parties, potentially after computing on this state (as is the case of MPC).

YOSO model: We say protocols with the aforementioned property are in the YOSO (*i.e.* You Only Speak Once) model, since parties are only required to act in a protocol execution when selected at random, which potentially only happens once. The YOSO model is especially interesting in

*Ignacio Cascudo was supported by the Spanish Government under the project SecuRing (ref. PID2019-110873RJ-I00) and by a research grant from Nomadic Labs and the Tezos Foundation.

†Bernardo David was supported by the Concordium Foundation and by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B.

‡Lydia Garms was supported by a research grant from Nomadic Labs and the Tezos Foundation.

§Anders Konring was supported by the IRFD grant number 9040-00399B (TrA^{2}C).

massively distributed settings (*e.g.* blockchains), where a huge number of parties are potentially involved but it is desirable to have only smaller committees execute a protocol for the sake of efficiency. Using small committees saves computation and communication, and since the identity of parties in the committee currently holding secret states is not known, an adversary cannot do better than corrupt random parties.

Role Assignment: At the core of protocols in the YOSO model is a scheme for encrypting messages towards *roles* rather than parties. A party randomly selected to perform a role can decrypt the messages sent to that role. This allows for executing traditional secret sharing [2] or MPC [16] protocols among roles that are performed by different parties as the execution proceeds. Besides passing confidential messages among parties assigned to certain roles, it is also paramount to allow parties to authenticate outgoing messages on behalf of the role they have just performed. This task has been modeled [16] and realized [2,18] as a functionality that outputs public keys for a random subset of anonymous parties in such a way that these parties can both decrypt messages encrypted under these keys and prove they were the rightful receivers. However, existing methods for role assignment [2,18,7] are still based on powerful primitives (*e.g.* FHE), incur too high costs and, most importantly, are incompatible with efficient techniques for publicly proving that encrypted secret shares are valid.

In this work we design schemes for role assignment that are not only efficient in sending messages to parties selected in the future but also amenable to the currently best techniques for publicly proving that encrypted messages are valid shares of a secret state, which is central to protocols in the YOSO model.

1.1 Related Works

Keeping Secrets: The seminal solution of [2] starts by selecting an auxiliary committee via an anonymous lottery (*e.g.* based on a VRF). Each party in this committee generates an ephemeral key pair and publishes the ephemeral public key and an encryption of the ephemeral secret key under the long-term public key of a party they choose at random. Encrypting towards an anonymous party can be done by encrypting under its ephemeral public key. However, since corrupted parties in the auxiliary committee will always choose other corrupted parties while the honest parties choose at random, this method needs a corruption ratio of 1/4 of the parties in order to arrive at an honest majority committee.

RPIR: The constraint on corruption ratio of [2] was subsequently solved in [18] via random-index private information retrieval (RPIR). RPIR allows a client to retrieve a random index from a database in such a way that the servers holding the database do not learn what index was retrieved. The solution of [18] consists in running a RPIR protocol with a database holding the public keys of all parties and having parties in a committee execute the client using MPC, outputting re-randomized versions of the public keys output by RPIR. While this solution allows for working in an honest majority scenario and achieves better asymptotic efficiency than [2], the concrete complexity is still quite high.

Encryption to the Future: A different approach is taken in [7], which constructs a primitive called Encryption to the Future (ETF). Instead of having committees actively participate in selecting future committees and help them receive their messages, ETF allows for non-interactively encrypting towards the winner of a lottery that is executed as part of an underlying blockchain ledger. Also, it allows for a party to prove it was the winner of this lottery (*i.e.* the receiver of a ciphertext) without exposing whether it won future lotteries. Although this solution can be constructed from simple tools like garbled circuits and oblivious transfer (after a setup phase), each encryption still requires communication and computational complexities linear in the total number of parties.

The ETF construction of [7] relies on a relaxation of Witness Encryption called Witness Encryption over Commitments (cWE), where one can encrypt a message towards the holder of an opening of a commitment to a valid witness of an NP relation. More specifically, we are interested in the case of Encryption to the Current Winner (ECW), where the data needed to determine the party selected to perform a role is already in the underlying blockchain (but still does not reveal who the party is). In order to realize ECW, each party commits to a witness of a predicate showing they win a lottery for the current parameter. A party encrypting towards a role simply encrypts the message towards the party who has such a committed witness to winning the lottery for a current parameter. A party who wins can decrypt the message encrypted towards the role using their witness. They can perform *Authentication from the Past* (AfP) on a message by doing a signature of knowledge on that message using their lottery winning witness.

The ETF constructions of [7] suffer from a major drawback: every encryption towards an anonymously selected party has communication complexity $O(n\kappa)$ where n is the *total* number of parties and κ is the security parameter. Even if preprocessing is allowed, these constructions still require the sender to publish n cWE ciphertexts or to have the eligible receivers perform a round of anonymous broadcast that is only usable for a single encryption. On the other hand, the AfP constructions only have $O(\kappa)$ communication complexity.

PVSS Compatibility: A drawback in current role assignment [2,18,7] is that they are not amenable to publicly verifiable secret (re)sharing. Both in YOSO proactive secret sharing [2] and YOSO MPC [16], the committees executing each round of the protocol do not simply send unstructured messages but shares of a secret that must be verified. While this can be done via generic non-interactive zero knowledge proofs of encrypted shares validity, such a solution incurs very high computational and communication costs.

Publicly Verifiable Secret Sharing (PVSS): An integral part of YOSO protocols is having each committee perform PVSS towards the next committee. A PVSS scheme allows for any party to check that an encrypted share vector is valid. A number of PVSS constructions are known [28,12,27,3,25,20] that different techniques for proving that a vector of encrypted shares are valid shares of a given secret. Recently, the SCRAPE [8] and ALBATROSS [9] PVSS schemes have significantly improved on the complexity of such schemes by making the share validity check and reconstructions procedures cheaper than previous works. While these works are based on number theoretical assumptions, a recent work has shown how to efficiently build PVSS from lattice based assumptions [17]. These works are not fit for the YOSO model because they require the parties to know the identities (or rather the public keys) of the parties receiving the shares when checking share validity, precluding (re)sharing towards anonymous parties. A key part of this work is that we explore the fact that the share validity check of SCRAPE can be modified to work regardless of the public keys used to encrypt the shares.

1.2 Our Contributions

In this work we address the issue of constructing simple ECW schemes amenable to efficient publicly verifiable secret (re)sharing (PVSS) protocols. Our contributions are summarized as follows:

Simple Encryption to Future (ECW): We construct a simple ECW scheme based on a mixnet and an additively homomorphic public key encryption scheme. Our scheme requires a setup phase where a mixnet is used but this setup can be either done once and reused for multiple times (using our reusable AFP) or preprocessed so that future encryptions can be done non-interactively. Our ECW ciphertexts have size linear *only in the number of parties who open them.*

Reusable Private Authentication from the Past (AFP): We show how to reuse our ECW setup even when a party performs multiple rounds of AFP, *i.e.* proving that it was selected to decrypt a given ECW ciphertext. This scheme guarantees that the adversary cannot predict which parties can decrypt future ECW ciphertexts while keeping the setup constant size.

Generic Efficient PVSS: We construct a generic PVSS protocol with efficient proofs of encrypted shares validity from any IND-CPA additively homomorphic encryption scheme with an efficient proof of decryption correctness without any generic zero knowledge proofs. This general result sheds new light on the construction on efficient PVSS schemes.

Efficient PVSS for Anonymous Committees based on ECW: We instantiate our generic PVSS construction based on our ECW and AFP schemes along with a protocol for resharing a secret towards a future random anonymous committee. This allows for parties to keep a secret alive, which is a core component of YOSO MPC.

1.3 Our Techniques

In this section we highlight the main technical components of our contributions. We remark that our main goal is providing simple constructions that yield efficient instantiations of PVSS towards anonymous committees along with efficient AfP schemes allowing parties to prove they received shares sent to a given role.

Encryption to the Future We introduce a simple ECW protocol where each party chooses a key pair in the system and then a mixnet is used to anonymize them. We can then define a simple lottery predicate that selects one of these keys. The winner of the lottery can trivially know that they have won this lottery. By combining this with an IND-CPA encryption scheme that encrypts a message under that key, we can obtain IND-CPA ECW. Using a homomorphic encryption scheme we can also encrypt to multiple lottery winners and prove that the same message is received by all of them.

Authentication from the Past

The Easy Way: An easy way of obtaining reusable ECW setup is to repeat the lottery setup and obtain multiple anonymized keys for each party. Then, any party can use a new anonymized public key for each AFP tag. This ensures that the AFP scheme can be executed a bounded number times before lottery winners can be linked to specific public keys in the setup and ciphertexts starts betraying their receivers.

The Reusable Way: In Appendix D, we show that a party can prove membership in a given committee without needing to reveal its role in this committee. This is done by signing a message with a ring signature [24] where the secret key corresponds to a public key in the committee. These signatures hide the identity of the party. Moreover, we require the signature to be linkable [21], so that no two parties can claim the same secret key. Using this and an anonymous channel, we can construct an AfP that can be used multiple times without linking a party P_i to its setup public key. More interestingly, we also present a protocol that leverages the presence of a dealer (which could be a party that encrypted the message to that committee) to reduce the size of these proofs of membership to constant (for the parties making the claims). This uses Camenisch-Lysyanskaya signatures[6], where the dealer signs the public keys of the committee, and the parties can then "complete" one of these signatures without revealing which one. We introduce a simple linkable version of these signatures.

PVSS We introduce two constructions for PVSS. The first, HEPVSS, is based on a generic encryption scheme which enjoys certain linearity properties with respect to encryption and decryption, and has the advantage that the security of the PVSS can be based on IND-CPA security of the scheme. The homomorphic properties of the scheme allow for simple proofs of sharing correctness and reconstruction. While we are only aware of El Gamal scheme satisfying the notion of the homomorphic properties we need, we hope this abstraction allows to capture other encryption schemes with homomorphic properties such as latticed-based assumption or Paillier in future work. In our second scheme DHPVSS, we introduce the idea of providing the dealer with an additional secret key for share distribution. This idea is powerful in combination with a technique used in SCRAPE to prove that encrypted shares lie on a polynomial of the right degree. The point is that, while in SCRAPE, this needed to be accompanied by a discrete logarithm equality proof for each share, now we only need one such proof. This reduces the sharing correctness proof to only 1 group element and a $\mathbb{Z}_p$-element, which essentially halves the communication complexity with respect to SCRAPE. We also introduce PVSS resharing protocols for both constructions, where a committee, among which a secret is PVSSed, can create shares of the same secret for the next committee, in a publicly verifiable way.

PVSS Towards Anonymous Committees Finally, we show that we can replace standard encryption and authentication in our PVSS protocols by ECW and AFP and thereby obtain PVSS toward anonymous committees.

2 Preliminaries

2.1 Sigma-protocols

At several points of this paper we will require non-interactive zero knowledge arguments of knowledge. We will implement these as Fiat-Shamir transformations of Σ-protocols. In fact, most of our statements are instances of the following general structure: let $\mathbb{F}$ be a finite field and let be $\mathcal{W}$ and $\mathcal{X}$ be $\mathbb{F}$-vector spaces, let $f : \mathcal{W} \to \mathcal{X}$ be a vector space homomorphism. Given $x \in \mathcal{X}$, we want to construct a proof of knowledge of witness $w \in \mathcal{W}$ for the relation

$$R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}.$$

In these conditions, the standard Σ-protocol π_{Pre} for this relation is as in Figure 1. It is well known that Π_{Pre} is a zero knowledge proof of knowledge with soundness error $1/|\mathbb{F}|$ (see Appendix E.1). The corresponding non-interactive Fiat-Shamir transform is as in Figure 2 and is a non-interactive zero-knowledge (NIZK) proof of knowledge in the random oracle model.

Generic Σ-protocol $\Pi_{\mathsf{Pre}}(w; x, f)$

Proof of knowledge of witness w for x with respect to the relation $R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}$, where $\mathcal{W}, \mathcal{X}$ are vector spaces over a finite field $\mathbb{F}$ and $f : \mathcal{W} \to \mathcal{X}$ is a $\mathbb{F}$-vector homomorphism.

Protocol:

1. The prover samples $r \leftarrow\!\$\, \mathcal{W}$, sends $a = f(r)$ to the verifier.
2. The verifier samples $e \leftarrow\!\$\, \mathbb{F}$, sends it to the sender.
3. The prover sends $z \leftarrow r + e \cdot w$ to the verifier.
4. The verifier accepts if $z \in \mathcal{W}$ and $f(z) = a + e \cdot x$.

Fig. 1. Generic Σ-protocol for knowledge of homomorphism-preimage

Generic non-interactive argument of knowledge $\Pi_{\mathsf{NI-Pre}}(w; x, f)$

Non-interactive argument of knowledge of witness for x for the relation $R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}$ in the random oracle model.

Public parameters: Finite field $\mathbb{F}$, vector spaces $\mathcal{W}, \mathcal{X}$ over $\mathbb{F}$, vector space homomorphism $f : \mathcal{W} \to \mathcal{X}$, $x \in \mathcal{X}$, random oracle $\mathcal{H} : \{0, 1\}^* \to \mathbb{F}$. Let $pp = (\mathbb{F}, \mathcal{W}, \mathcal{X}, \mathcal{H})$.

Algorithm 1 $\Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$

$r \leftarrow\!\$\, \mathcal{W}$
$a \leftarrow f(r)$, $e \leftarrow \mathcal{H}(x, a)$, $z \leftarrow a + e \cdot x$
return $\pi \leftarrow (e, z)$

Algorithm 2 $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp, x, f, \pi)$

Parse $\pi = (e, z)$
$a \leftarrow f(z) - e \cdot x$
return accept if and only if
$z \in \mathcal{W}$ and $e = \mathcal{H}(x, a)$.

Fig. 2. Generic non-interactive argument of knowledge of homomorphism-preimage

Cyclic group homomorphism preimage, DL knowledge and DLEQ knowledge proofs A cyclic group $\mathbb{G}$ of prime order p has a vector space structure over the field $\mathbb{Z}_p$, and a group homomorphism $f : \mathbb{G} \to \mathbb{G}'$ between groups of order p is also a $\mathbb{Z}_p$-vector homomorphism.[3]

Some examples of homomorphism-preimage proofs are proofs of knowledge of discrete logarithm and discrete logarithm equality.

Let G be a generator of $\mathbb{G}$. Given X a discrete logarithm DL proof of knowledge $\mathsf{DL}(w; G, X)$ asserts knowledge of $w \in \mathbb{Z}_p$ with $X = w \cdot G$ (we denote this as $w = \mathsf{DL}_G(X)$). In the language above this is provided by $\Pi_{\mathsf{NI-Pre}}(w; (X), f_G)$ with $f_G(w) = w \cdot G$. This is the non-interactive version of the well known Schnorr proof.

Similarly, let G, H be elements in $\mathbb{G}$. Given $X, Y \in \mathbb{G}$ we will call $\mathsf{DLEQ}(w; G, X, H, Y)$ (discrete logarithm proof) a non-interactive proof of knowledge of $w \in \mathbb{Z}_p$ with $w = \mathsf{DL}_G(X) = \mathsf{DL}_H(Y)$, which can be obtained by using $\Pi_{\mathsf{NI-Pre}}(w; (X, Y), f_{(G,H)})$, where $f_{G,H}(w) = (w \cdot G, w \cdot H)$.

2.2 $\mathbb{Z}_p$-linear Homomorphic Encryption

The results in this paper require encryption schemes with certain homomorphic properties, that allow for simple proofs of plaintext knowledge. These are admittedly restrictive, and we are only aware of them being attained by El Gamal encryption scheme (described in Appendix A); neverthe-

[3] This extends to direct products of groups of order p, i.e. $\mathcal{W} = \mathbb{G}_1 \times \cdots \times \mathbb{G}_m$, $\mathcal{X} = \mathbb{G}'_1 \times \cdots \times \mathbb{G}'_n$ and $f = (f_1, \ldots, f_m) : \mathcal{W} \to \mathcal{X}$ where $f_i : \mathbb{G}_i \to \mathcal{X}$ are all group homomorphisms.

less we introduce the following notion in order to capture the precise algebraic properties we need, and in the hope this can make it easier to generalize our techniques to other encryption schemes.

Definition 1 ($\mathbb{Z}_p$-linearly homomorphic encryption scheme). *Let $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ be a public key encryption scheme (see Appendix A for a definition), and let p be a prime number. We say $\mathcal{E}$ is $\mathbb{Z}_p$-linearly homomorphic ($\mathbb{Z}_p$-LHE) if the plaintext space $(\mathfrak{P}, \boxplus_{\mathfrak{P}})$, randomness space $(\mathfrak{R}, \boxplus_{\mathfrak{R}})$, ciphertext space $(\mathfrak{C}, \boxplus_{\mathfrak{C}})$ each have a $\mathbb{Z}_p$-vector space structure and for all public keys pk output by $\mathcal{E}.\mathsf{Gen}$, $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{C}$ is a $\mathbb{Z}_p$-vector space homomorphism, i.e. for all $m_1, m_2 \in \mathfrak{C}$, $\rho_1, \rho_2 \in \mathfrak{R}$,*

$$\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m_1; \rho_1) \boxplus_{\mathfrak{C}} \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m_2; \rho_2) = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m_1 \boxplus_{\mathfrak{P}} m_2; \rho_1 \boxplus_{\mathfrak{R}} \rho_2).$$

Proofs of decryption correctness $\mathbb{Z}_p$-linear homomorphic encryption schemes have simple proofs of knowledge of plaintext message, since these can be cast as proofs of homomorphic preimage. We will use this fact at various places.

However, we will also need proofs of decryption correctness, where of course the prover wants to keep their secret key hidden, i.e. proofs for the relation

$$R_{\mathcal{E},\mathsf{Dec}} = \{(\mathsf{sk}; (\mathsf{pk}, m, c)) : (\mathsf{pk}, \mathsf{sk}) \text{ is a valid key-pair for } \mathcal{E} \text{ and } m = \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(c)\}.$$

If the prover knows the randomness under which the message was encrypted, the proving algorithm $\mathcal{E}.\mathsf{ProveDec}(\mathsf{sk}; (\mathsf{pk}, m, c))$ can simply output that randomness $\pi \in \mathfrak{R}$; the verification $\mathcal{E}.\mathsf{VerifyDec}(\mathsf{pk}, m, c, \pi)$ accepts if $\mathsf{Enc}_{\mathsf{pk}}(m; \pi) = c$.

Unfortunately El Gamal encryption scheme does not allow a decryptor to retrieve the randomness under which a message has been encrypted. Instead, a proof of correctness of decryption for El Gamal can be constructed from the following property of this scheme, which we call $\mathbb{Z}_p$-linear decryption.

Definition 2. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a $\mathbb{Z}_p$-linearly homomorphic encryption scheme and denote $\mathcal{PK}$ and $\mathcal{SK}$ the sets of public and secret keys respectively. We say $\mathcal{E}$ has $\mathbb{Z}_p$-linear decryption if:*

- *$\mathcal{PK}$ and $\mathcal{SK}$ are $\mathbb{Z}_p$-vector spaces.*
- *There exists a $\mathbb{Z}_p$-linear homomorphism $F : \mathcal{SK} \to \mathcal{PK}$ such that $\mathsf{pk} = F(\mathsf{sk})$ for all $(\mathsf{pk}, \mathsf{sk})$ outputted by Gen.*
- *For all $c \in \mathfrak{C}$, the function $D_c(\mathsf{sk}) := \mathsf{Dec}_{\mathsf{sk}}(c)$ is $\mathbb{Z}_p$-linear in sk, i.e. for all $\mathsf{sk}_1, \mathsf{sk}_2 \in \mathcal{SK}$, it holds that $D_c(\mathsf{sk}_1 \boxplus_{\mathcal{SK}} \mathsf{sk}_2) = D_c(\mathsf{sk}_1) \boxplus_{\mathfrak{P}} D_c(\mathsf{sk}_2)$.*

In this case we have the algorithms ($\mathcal{E}.\mathsf{ProveDec}$, $\mathcal{E}.\mathsf{VerifyDec}$) that constitute a NIZK proof for $R_{\mathcal{E},\mathsf{Dec}}$:

Algorithm 3 $\mathcal{E}.\mathsf{ProveDec}(\mathsf{sk}, (\mathsf{pk}, m, c))$

$\mathcal{W} \leftarrow \mathcal{SK}, \mathcal{X} \leftarrow \mathcal{PK} \times \mathfrak{P} \times \mathfrak{C},$
$pp \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
$w \leftarrow \mathsf{sk},\ x \leftarrow (\mathsf{pk}, m),\ f(\cdot) \leftarrow (F(\cdot), D_c(\cdot))$
return $\pi \leftarrow \mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$

Algorithm 4 $\mathcal{E}.\mathsf{VerifyDec}(\mathsf{pk}, m, c, \pi)$

$\mathcal{W} \leftarrow \mathcal{SK}, \mathcal{X} \leftarrow \mathcal{PK} \times \mathfrak{P} \times \mathfrak{C}$
$pp \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
$x \leftarrow (\mathsf{pk}, m),\ f(\cdot) \leftarrow (F(\cdot), D_c(\cdot))$
return $\mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp, x, f)$

The El Gamal decryption function as usually described is not linear but affine, but we can easily fix this by e.g. defining $\mathsf{sk}^* = (\mathsf{sk}_1^*, \mathsf{sk}_2^*) = (1, \mathsf{sk}) \in \mathbb{Z}_p^2$ and letting $\mathsf{Dec}_{\mathsf{sk}^*}(C_1, C_2) := C_2 \cdot \mathsf{sk}_1^* - C_1 \cdot \mathsf{sk}_2^*$. Then $D_C(\mathsf{sk}^*)$ is clearly a $\mathbb{Z}_p$-linear function.

2.3 Shamir Secret Sharing on Groups of Order p

The well known degree-t Shamir scheme allows to split a secret $s \in \mathbb{Z}_p$ in n shares (where $0 \leq t < n < p$) in such a way that any set of $t+1$ shares give full information about the secret s while any set of t give no information on s.

However, we will consider situations where the secret is an element $S = sG$ of a group $\mathbb{G}$ of order p with generator G, but the dealer does not know s (and hence cannot apply the usual Shamir sharing using s as secret). On the other hand, it will be enough that the shares allow to reconstruct S and not s. We define Shamir secret sharing in a group of order p as shown in Figure 3.(Shamir secret sharing scheme over $\mathbb{Z}_p$ is retrieved by setting $\mathbb{G} = (\mathbb{Z}_p, +)$, $G = 1$). We denote by $\mathbb{Z}_p[X]_{\leq t}$ the set of polynomials in $\mathbb{Z}_p[X]$ of degree at most t.

Shamir secret sharing on a group $\mathbb{G}$ of order p

Public parameters: Let $pp = (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\})$, where $\mathbb{G}$ is a group of prime order p with generator G, $0 \leq t < n < p$ are integers, and $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$ are pairwise distinct.

Algorithm 5 $\mathsf{GShamir.Share}(pp, S)$

Input: $S \in \mathbb{G}$
$m(X) \leftarrow_\$ \{m(X) \in \mathbb{Z}_p[X]_{\leq t} : m(\alpha_0) = 0\}$
$A_i = S + m(\alpha_i) \cdot G,\ i \in [n]$
return $(A_1, \ldots, A_n)$

Algorithm 6 $\mathsf{GShamir.Rec}(pp, I, \{A_i\}_{i \in I})$

Input: $I \subseteq [n], |I| = t+1, \{A_i\}_{i \in I}$
$\forall i \in I, \quad \lambda_{i,I} \leftarrow \prod_{j \in I, j \neq i} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j}$
$S' \leftarrow \sum_{i \in I} \lambda_{i,I} A_i$
return S'

Fig. 3. Shamir sharing on a group of order p

2.4 The SCRAPE Test

In [8], a technique for checking correctness of Shamir sharing in publicly verifiable secret sharing was introduced. Letting aside the details on how the technique works in SCRAPE, we are interested in the following fact, which in turn comes from well known results in coding theory [4].

Theorem 1 (SCRAPE dual-code test). *Let $1 \leq t < n$ be integers. Let p be a prime number with $p \geq n$. Let $\alpha_1, \ldots, \alpha_n$ be pairwise different points in $\mathbb{Z}_p$. Define the coefficients $v_i = \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}$. Let*

$$C = \{(m(\alpha_1), \ldots, m(\alpha_n)) : \ m(X) \in \mathbb{Z}_p[X]_{\leq t}\}.$$

Then for every vector $(\sigma_1, \ldots, \sigma_n)$ in $\mathbb{Z}_p^n$,

$$(\sigma_1, \ldots, \sigma_n) \in C \quad \Leftrightarrow \quad \sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot \sigma_i = 0, \quad \forall m^* \in \mathbb{Z}_p[X]_{\leq n-t-1}.$$

2.5 Mix Networks (Mixnets)

In this paper we use a mixnet to anonymize a set of public encryption keys, each generated (with their corresponding secret keys) by a party in the system. Let $\mathcal{P}$ be the set of all parties generating

[4] Specifically from the fact that the dual of a Reed-Solomon code is a generalized Reed-Solomon code of a certain form

these keys. In the coming sections we will assume such a mixnet and that the output is subsequently be written to a blockchain. The output is a set of shuffled keys $\mathsf{pk}_{\mathsf{Anon},j} : j \in [n]$, for which each party knows the index that corresponds to their public key, but nothing else about the permutation. Denote this permutation $\psi : \mathcal{P} \to [n]$, i.e. party ID_i knows $j = \psi(i)$ and the corresponding key-pair. We will use the fact that a party can encrypt a message under the public key $\mathsf{pk}_{\mathsf{Anon},j}$. It is clear that party $ID_{\psi^{-1}(j)}$ can decrypt the message, while the rest of the parties (even the sender) remain oblivious about the identity of the receiver. Notice that this setup can be instantiated via a verifiable mixnet (*e.g.* [4]).

2.6 Encryption to the Future

We use the model for Encryption to the Future (EtF) from [7], which defines this primitive with respect to a blockchain ledger that has an built-in lottery mechanism. Before presenting the definition of EtF and related concepts, we recall the model for blockchain ledgers from [19], which is used to state the definitions of [7] and that captures properties of natural Proof-of-Stake (PoS) based protocols such as [11]. We present a summary of the framework in Appendix B and discuss below the main properties we will use in the EtF definitions.

Blockchain Structure A genesis block
$B_0 = (\mathsf{Sig.pk}_1, \mathsf{aux}_1, \mathsf{stake}_1), \ldots, (\mathsf{Sig.pk}_n, \mathsf{aux}_n, \mathsf{stake}_n), \mathsf{aux}$ associates each party P_i to a signature scheme public key $\mathsf{Sig.pk}_i$, an amount of stake stake_i and auxiliary information aux_i (*i.e.* any other relevant information required by the blockchain protocol). As in [11], we assume that the genesis block is generated by an initialization functionality $\mathcal{F}_{\mathsf{INIT}}$ that registers all parties' $\mathsf{Sig.pk}_i, \mathsf{aux}_i$ when the execution starts and assigns stake_i for P_i. Within the execution model of [19], $\mathcal{F}_{\mathsf{INIT}}$ is executed by the environment (as defined in Appendix B). A blockchain $\mathbf{B}$ relative to a genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence of slots $\mathsf{sl}_1, \ldots, \mathsf{sl}_m$ such that $B_i = (\mathsf{sl}_j, H(B_{i-1}), \mathsf{d}, \mathsf{aux})$, where sl_j indicates the time slot that B_i occupies, $H(B_{i-1})$ is a collision resistant hash of the previous block, d is data and aux is auxiliary information required by the blockchain protocol (*e.g.* a proof that the block is valid for slot sl_j). We denote by $\mathbf{B}^{\lceil \ell}$ the chain (sequence of blocks) $\mathbf{B}$ where the last ℓ blocks have been removed and if $\ell \geq |\mathbf{B}|$ then $\mathbf{B}^{\lceil \ell} = \epsilon$. Also, if $\mathbf{B}_1$ is a prefix of $\mathbf{B}_2$ we write $\mathbf{B}_1 \preceq \mathbf{B}_2$. For the sake of simplicity, we identify each party P_i participating in the protocol by its public key $\mathsf{Sig.pk}_i$.

Evolving Blockchains In an EtF scheme, the future is defined with respect to a future state of the underlying blockchain. In particular, we want to make sure that the initial chain $\mathbf{B}$ has "correctly" evolved into the final chain $\tilde{\mathbf{B}}$. Otherwise, the adversary can easily simulate a blockchain where it wins a future lottery and finds itself with the ability to decrypt. Fortunately, the *Distinguishable Forking* property from [19] allows us to distinguish a sufficiently long chain in an honest execution from a fork generated by the adversary by looking at the combined amount of stake proven in such a sequence of blocks. This property is used to construct a predicate called $\mathsf{evolved}(\cdot, \cdot)$. First, let $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol with validity predicate V and where the $(\alpha, \beta, \ell_1, \ell_2)$-*distinguishable forking* property holds. And let $\mathbf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$ and $\tilde{\mathbf{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$.

Definition 3 (Evolved Predicate). *An evolved predicate is a polynomial time function* $\mathsf{evolved}$ *that takes as input blockchains* $\mathbf{B}$ *and* $\tilde{\mathbf{B}}$

$$\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) \in \{0, 1\}.$$

It outputs 1 if and only if $\mathbf{B} = \tilde{\mathbf{B}}$ *or the following holds (i)* $V(\mathbf{B}) = V(\tilde{\mathbf{B}}) = 1$*; (ii)* $\mathbf{B}$ *and* $\tilde{\mathbf{B}}$ *are consistent i.e.* $\mathbf{B}^{\lceil \kappa} \preceq \tilde{\mathbf{B}}$ *where* κ *is the common prefix parameter; (iii) Let* $\ell' = |\tilde{\mathbf{B}}| - |\mathbf{B}|$ *then it holds that* $\ell' \geq \ell_1 + \ell_2$ *and* $\mathsf{u\text{-}stakefrac}(\tilde{\mathbf{B}}, \ell' - \ell_1) > \beta$*.*

Blockchain Lotteries The vast majority of PoS-based blockchain protocols has an inbuilt lottery scheme for selecting parties to generate blocks. In this lottery any party can win the right to generate a block for a certain slot with a probability proportional to its relative stake in the system. In the model from [7], a party can decrypt an EtF ciphertext if it wins this lottery. It can be useful to conduct multiple independent lotteries for the same slot sl, which is associated to a set of roles $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Depending on the lottery mechanism, each pair $(\mathsf{sl}, \mathsf{P}_i)$ may yield zero, one or multiple winners. A party with access to the blockchain can locally determine whether it is the lottery winner for a given role by executing a procedure using its lottery witness $\mathsf{sk}_{L,i}$ related to $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$, which may also give the party a proof of winning for others to verify. The definition below from [7] details what it means for a party to win a lottery.

Definition 4 (Lottery Predicate). *A lottery predicate is a polynomial time function* $\mathsf{lottery}$ *that takes as input a blockchain* $\mathbf{B}$*, a slot* sl*, a role* P *and a lottery witness* $\mathsf{sk}_{L,i}$ *and outputs 1 if and only if the party owning* $\mathsf{sk}_{L,i}$ *won the lottery for the role* P *in slot* sl *with respect to the blockchain* $\mathbf{B}$*.*
Formally, we write $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) \in \{0, 1\}$*.*

It is natural to establish the set of lottery winning keys $\mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{P}}$ for parameters $(\mathbf{B}, \mathsf{sl}, \mathsf{P})$. This is the set of eligible keys satisfying the $\mathsf{lottery}$ predicate.

Modelling EtF. We are now ready to present the model of [7] for encryption to the future winner of a lottery (*i.e.* EtF). The blocks of an underlying blockchain ledger and their relative positions in the chain are used to specify points in time. Intuitively, this notion allows for creating ciphertexts that can only be decrypted by a party that is selected to perform a certain role R at a future slot sl according to a lottery scheme associated with a blockchain protocol (*i.e.* a party that has a lottery secret key $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$).

Definition 5 (Encryption to the Future). *A pair of PPT algorithms* $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ *in the the context of a blockchain* Γ^V *is an EtF-scheme with evolved predicate* $\mathsf{evolved}$ *and a lottery predicate* $\mathsf{lottery}$*. The algorithms work as follows*

Encryption. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m)$ *takes as input an initial blockchain* $\mathbf{B}$*, a slot* sl*, a role* P *and a message* m*. It outputs a ciphertext* ct *- an encryption to the future.*

Decryption. $m/\bot \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk})$ *takes as input a blockchain state* $\tilde{\mathbf{B}}$*, a ciphertext* ct *and a secret key* sk *and outputs the original message* m *or* $\bot$*.*

Correctness. An EtF-scheme is said to be correct if for honest parties i and j, there exists a negligible function μ such that

$$\left| \Pr \left[\begin{array}{c} \mathsf{view} \leftarrow \mathit{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \mathbf{B} = \mathsf{GetRecords}(\mathsf{view}_i) \\ \tilde{\mathbf{B}} = \mathsf{GetRecords}(\mathsf{view}_j) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m) \\ \mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1 \\ \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}) = 1 \end{array} : \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk}) = m \right] - 1 \right| \leq \mu(\lambda).$$

Security. Security is defined with a game $\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ described in Algorithm 7, where a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ execute an underlying blockchain protocol with an environment $\mathcal{Z}$ as described in Appendix B. In this game, $\mathcal{A}$ chooses a blockchain $\mathbf{B}$, a role P for the slot sl and two messages m_0 and m_1 and sends it all to $\mathcal{C}$, who chooses a random bit b and encrypts the message m_b with the parameters it received and sends ct to $\mathcal{A}$. $\mathcal{A}$ continues to execute the blockchain until an evolved blockchain $\tilde{\mathbf{B}}$ is obtained and outputs a bit b'. If the adversary is a lottery winner for the challenge role P in slot sl, the game outputs a random bit. If the adversary is not a lottery winner for the challenge role P in slot sl, the game outputs $b \oplus b'$. The reason for outputting a random guess in the game when the challenge role is corrupted is as follows. Normally the output of the IND-CPA game is $b \oplus b'$ and we require it to be 1 with probability 1/2. This models that the guess b' is independent of b. This, of course, cannot be the case when the challenge role is corrupted. We therefore output a random guess in these cases. After this, any bias of the output away from 1/2 still comes from b' being dependent on b.

Algorithm 7 $\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$

```
view^r ← EXEC_r^Γ(A, Z, 1^λ)                          ▷ A executes Γ with Z until round r
(B, sl, P, m_0, m_1) ← A(view^r_A)                     ▷ A outputs challenge parameters
b ←$ {0,1}
ct ← Enc(B, sl, P, m_b)
st ← A(view^r_A, ct)                                   ▷ A receives challenge ct
view^r̃ ← EXEC^Γ_(view^r, r̃)(A, Z, 1^λ)                ▷ Execute from view^r until round r̃
(B̃, b') ← A(view^r̃_A, st)
if evolved(B, B̃) = 1 then                              ▷ B̃ is a valid evolution of B
    if sk^A_{L,j} ∉ W_{B̃,sl,P} then                    ▷ A does not win role P
        return b ⊕ b'
    end if
end if
return b̂ ←$ {0,1}
```

Definition 6 (IND-CPA Secure EtF). *An EtF-scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ in the context of a blockchain protocol Γ executed by PPT machines $\mathcal{A}$ and $\mathcal{Z}$ is said to be* IND-CPA *secure if, for any $\mathcal{A}$ and $\mathcal{Z}$, there exists a negligible function μ such that for $\lambda \in \mathbb{N}$:*

$$\left| 2 \cdot \Pr \left[\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}} = 1 \right] - 1 \right| \leq \mu(\lambda).$$

ECW as a Special Case of EtF. In this work, we focus on a special class of EtF called ECW where the underlying lottery is always conducted with respect to the current blockchain state. This has the following consequences

1. $\mathbf{B} = \tilde{\mathbf{B}}$ means that $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ is trivially true.
2. The winner of role P in slot sl is already defined in $\mathbf{B}$.

Notice that in ECW there is no need for checking if the blockchain has 'correctly' evolved and all lottery parameters (*e.g.* stake distribution and randomness extracted from the blockchain) are static. Hence, when constructing an ECW scheme, the lottery winner is already decided at encryption time. While an ECW is simpler to realize than a more general EtF, it is shown in [7] that ECW can be used to instantiate YOSO MPC and then be transformed into EtF given an identity based encryption scheme.

Authentication from the Past (AfP) When the winner of a role S sends a message m to a future role R then it is typically also needed that R can be sure that the message m came from a party P which, indeed, won the role S. This concept is formalized as an AfP scheme as follows.

Definition 7 (Authentication from the Past). *A pair of PPT algorithms $\mathcal{U} = (\mathsf{Sign}, \mathsf{Ver})$ is a scheme for authenticating messages as a winner of a lottery in the past in the context of blockchain Γ with lottery predicate* $\mathsf{lottery}$ *such that:*

Authenticate. $\sigma \leftarrow \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}, m)$ *takes as input a blockchain $\mathbf{B}$, a slot sl, a role S and a message m. It outputs a signature σ that authenticates the message m.*

Verify. $\{0,1\} \leftarrow \mathsf{AfP.Ver}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \sigma, m)$ *uses the blockchain $\tilde{\mathbf{B}}$ to ensure that σ is a signature on m produced by the secret key winning the lottery for slot sl and role S.*

Furthermore, an AfP-scheme has the following properties:

Correctness.

$$\left| \Pr\left[\begin{array}{c} \mathsf{view} \leftarrow \mathit{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \mathbf{B} = \mathsf{GetRecords}(\mathsf{view}_i) \\ \tilde{\mathbf{B}} = \mathsf{GetRecords}(\mathsf{view}_j) \\ \sigma \leftarrow \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}, m) \\ \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}) = 1 \\ \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}) = 1 \end{array} : \mathsf{AfP.Ver}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \sigma, m) = 1 \right] - 1 \right| \leq \mu(\lambda)$$

In other words, an AfP on a message from an honest party with a view of the blockchain $\mathbf{B}$ can attest to the fact that the sender won the role S in slot sl. If another party, with blockchain $\tilde{\mathbf{B}}$ agrees, then the verification algorithm will output 1.

Security. *The EUF-CMA game detailed in 8 is used to define the security of an AfP scheme. In this game, the adversary has access to a signing oracle $\mathcal{O}_{\mathsf{AfP}}$ which it can query with a slot sl, a role S and a message m_i, obtaining AfP signatures $\sigma_i = \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_j, m_i)$ where $\mathsf{sk}_j \in \mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{S}}$ i.e. $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_j) = 1$. The oracle maintains the list of queries $\mathcal{Q}_{\mathsf{AfP}}$. Formally, an AfP-scheme $\mathcal{U}$ is said to be EUF-CMA secure in the context of a blockchain protocol Γ executed by PPT machines $\mathcal{A}$ and $\mathcal{Z}$ if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$:*

$$\Pr\left[\mathrm{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}} = 1\right] \leq \mu(\lambda)$$

Algorithm 8 $\text{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}}$

```
view ← EXEC^Γ(A, Z, 1^λ)                                  ▷ A executes Γ with Z
(B, sl, S, m', σ') ← A^{O_AfP}(view_A)
if (m' ∈ Q_AfP) ∨ (sk^A_{L,j} ∈ W_{B,sl,S}) then          ▷ A^{O_AfP} won or queried illegal m'
    return 0
end if
view^r̃ ← EXEC^Γ_{(view^r, r̃)}(A, Z, 1^λ)                 ▷ Execute from view^r until round r̃
B̃ ← GetRecords(view^r̃_i)
if evolved(B, B̃) = 1 then
    if Ver(B, sl, S, σ', m') = 1 then                      ▷ A successfully forged an AfP
        return 1
    end if
end if
return 0
```

AfP Privacy The specific privacy property we seek is that an adversary, observing AfP tags from honest parties, cannot use this information to enhance its chances in predicting the winners of lotteries for roles for which an AfP tag has not been published.

Definition 8 (AfP Privacy.). *An AfP scheme $\mathcal{U}$ with corresponding lottery predicate* lottery *is private if a PPT adversary is unable to distinguish between the scenarios defined in 9 and 10 with more than negligible probablity in the security parameter.*

Scenario 0 ($b = 0$) *In this scenario (9) the adversary is first running the blockchain Γ together with the environment $\mathcal{Z}$. At round r the adversary is allowed to interact with the oracle $\mathcal{O}_{\mathsf{AfP}}$ as described in 7. The adversary then continues the execution until round $\tilde{r}$ where it ouputs a bit b'.*

Scenario 1 ($b = 1$) *This scenario (10) is identical to scenario 0 but instead of interacting with $\mathcal{O}_{\mathsf{AfP}}$, the adversary interacts with a simulator $\mathcal{S}$.*

Algorithm 9 $b = 0$

```
view^r ← EXEC^Γ_r(A, Z, 1^λ)
A^{O_AfP}(view^r_A)
view^r̃ ← EXEC^Γ_{(view^r, r̃)}(A, Z, 1^λ)
return b' ← A^{O_AfP}(view^r̃_A)
```

Algorithm 10 $b = 1$

```
view^r ← EXEC^Γ_r(A, Z, 1^λ)
A^S(view^r_A)
view^r̃ ← EXEC^Γ_{(view^r, r̃)}(A, Z, 1^λ)
return b' ← A^S(view^r̃_A)
```

We let $\text{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ denote the game where a coinflip decides whether the adversary is executed in scenario 0 or scenario 1. We say that the adversary wins the game (i.e. $\text{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}} = 1$) iff $b' = b$. Finally, an AfP scheme $\mathcal{U}$ is called private in the context of the blockchain Γ and underlying lottery predicate lottery *if the following holds for a negligible function μ.*

$$\Pr\left[\text{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}} = 1\right] \leq 1/2 + \mu(\lambda)$$

3 ECW based on $\mathbb{Z}_p$-Linearly Homomorphic Encryption

This section presents an ECW protocol based on a $\mathbb{Z}_p$-linearly homomorphic encryption scheme described in Section 2.2 and a mixnet (Section 2.5). Together with the ECW, we introduce an AfP scheme - a mechanism that allows a committee member to authenticate messages. The two schemes will be the backbone of the anonymous PVSS presented in Section 6. Before presenting the actual ECW and AfP protocols, we introduce the underlying lottery predicate that will be the cornerstone in our two schemes.

3.1 Lottery Predicate

We assume a running blockchain as described Section B and a function param that has access to the blockchain state. During the setup, each party samples an encryption key pair $(\mathsf{sk}_{\mathcal{E},i}, \mathsf{pk}_{\mathcal{E},i})$ and inputs $\mathsf{pk}_{\mathcal{E},i}$ to the mixnet (Section 2.5). The output of the mixnet is a tuple $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$ which is written on the blockchain and accessible to every party through param function. The function param takes as input the blockchain $\mathbf{B}$ and the slot sl and outputs a tuple $(\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}, \eta) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$. Here, $(j, \mathsf{pk}_{\mathsf{Anon},j})$ is equal to $(\psi(i), \mathsf{pk}_{\mathcal{E},i})$ for the permutation ψ defined by the mixnet. Finally, η is the public randomness from the blockchain corresponding to $\mathbf{B}$ and sl. Not, that only the owner of $\mathsf{sk}_{\mathcal{E},i}$ knows j such that $\mathsf{pk}_{\mathsf{Anon},j} = \mathsf{pk}_{\mathcal{E},i}$. Let $\mathcal{H} : \{0,1\}^* \rightarrow [n]$ be a hash function that outputs a number that points to a specific index in the list of public keys. The lottery predicate $\mathsf{lottery}$ is detailed below.

Algorithm 11 $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$

$(\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}, \eta) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$
$(\mathsf{pk}_{\mathcal{E},i}, \mathsf{sk}_{\mathcal{E},i}) \leftarrow \mathsf{sk}_{L,i}$
$k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{P}||\eta)$
return 1 iff $\mathsf{pk}_{\mathcal{E},i} = \mathsf{pk}_{\mathsf{Anon},k}$

It is easy to see that the lottery described above associates a *single* party (from the set of eligible parties) with the role P. Furthermore, the party can locally check if it won the lottery by checking that the output of the hash function points to its own public key in the permuted set. Crucially, the party winning the lottery can stay covert since no other party can link the winning lottery key to the owner of the corresponding secret key. These properties will be useful when we want to encrypt shares towards an anonymous committee.

3.2 ECW Protocol

This section introduces a ECW protocol (Figure 4) based on the lottery predicate presented in Section 3.1. We note that ECW is just a restricted version of EtF where the lottery is conducted wrt. the *current* blockchain $\mathbf{B}$ and slot sl. Thus, all definitions in Section 2.6 applies to ECW schemes too.

Theorem 2 (IND-CPA ECW). *Let $\mathcal{E}$ be an IND-CPA secure $\mathbb{Z}_p$-linearly homomorphic encryption scheme. The construction in Figure 4 with lottery predicate as in Section 3.1 is an IND-CPA secure ECW (as in Definition 6).*

(See proof sketch in Section C)

3.3 AfP Protocol

In this section we present our AfP protocol. It is described in Figure 5 and is based on a Signature of Knowledge (SoK) [10]. A SoK scheme is a pair of algorithms $(\mathsf{SoK.sign}, \mathsf{SoK.verify})$ and is defined in

ECW Protocol

Public parameters: A prime p, a $\mathbb{Z}_p$-linearly homomorphic encryption scheme $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ with notation as in Section 2.2 and a lottery as described in Section 3.1.

Set-up:
1. Every party runs $\mathcal{E}.\mathsf{Gen}()$ obtaining a key pair $(\mathsf{sk}_{\mathcal{E},i}, \mathsf{pk}_{\mathcal{E},i})$.
2. Each party inputs $\mathsf{pk}_{\mathcal{E},i}$ to the mixnet. The output of the mixnet is a tuple $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$ which is written on the blockchain and accessible to every party when using the param function.

Encryption protocol: Input $(\mathbf{B}, \mathsf{sl}, \mathsf{P})$ and $m \in \mathfrak{P}$.
1. Run $\mathsf{param}(\mathbf{B}, \mathsf{sl})$ and obtain $(\{(l, \mathsf{pk}_{\mathsf{Anon},l})\}_{l\in[n]}, \eta)$.
2. Obtain random index by $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{P}||\eta)$.
3. Choose ρ in $\mathfrak{R}$ and set $c = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{Anon},k}}(m, \rho)$.
4. Sender outputs c.

Decryption protocol: Input for party i is $\mathbf{B}, \mathsf{sk}_{L,i}$ and c.
1. Checks that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$.
2. Outputs $m = \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}_{\mathsf{Anon},i}}(c)$.

Fig. 4. ECW Protocol

context of a relation R. We consider statements of the form $x = (\mathbf{B}, \mathsf{sl}, \mathsf{P})$ and witnesses $w = \mathsf{sk}$. We say that $R(x = (\mathbf{B}, \mathsf{sl}, \mathsf{P}), w = \mathsf{sk}) = 1$ iff $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}) = 1$. A signature is produced by running $\sigma \leftarrow \mathsf{SoK.sign}(x, w, m)$. And it can be verified by checking that the output of $\mathsf{SoK.verify}(x, \sigma, m)$ is 1. Our AfP uses the SoK to sign m under the knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$. This will exactly attest that the message m was sent by the winner of the lottery for P. An instantiation of this AfP protocol could use DL proofs (Section 2.1).

AfP Protocol

Public parameters and **Set-up** as described in Figure 4 plus additional setup for the SoK scheme $\mathsf{SoK} = (\mathsf{SoK.sign}, \mathsf{SoK.verify})$.

Authentication protocol: Input for party i is $(\mathbf{B}, \mathsf{sl}, \mathsf{P})$ and $m \in \mathfrak{P}$.
1. Checks that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$.
2. Constructs an SoK on the message m of knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$ resulting in $\sigma \leftarrow \mathsf{SoK.sign}((\mathbf{B}, \mathsf{sl}, \mathsf{P}), \mathsf{sk}_{L,i})$.
3. Sender outputs $\sigma \leftarrow \sigma_{\mathsf{SoK}}$.

Verification protocol: Input is $(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \sigma, m)$
1. Parses σ as the SoK signature σ_{SoK}.
2. Verifies that σ_{SoK} is a valid SoK on the message m proving knowledge of $\mathsf{sk}_{L,i}$. I.e. it runs $b \leftarrow \mathsf{SoK.verify}((\mathbf{B}, \mathsf{sl}, \mathsf{P}), \sigma_{\mathsf{SoK}}, m)$.
3. Verifier outputs b.

Fig. 5. AfP Protocol

Theorem 3 (EUF-CMA AfP). *Let $\mathcal{E}$ be an IND-CPA secure and $\mathbb{Z}_p$-linearly homomorphic encryption scheme and let* SoK *be a simulatable and extractable SoK scheme. The construction in Figure 5 with lottery predicate as in Section 3.1 is EUF-CMA AfP as defined in Definition 7.*

(See proof sketch in Section C)

AfP Privacy The privacy property of an AfP scheme says that no adversary can distinguish between interacting with an AfP oracle $\mathcal{O}_{\mathsf{AfP}}$ and a simulator $\mathcal{S}$ during a blockchain execution. Intuitively, this provides the guarantee that observing other AfP tags does not enhance an adversary's chance of guessing future lottery winners.

Theorem 4 (AfP Privacy). *Assume $\mathcal{E}$,* $\mathsf{lottery}$ *and* SoK *scheme as in 6. The construction in Figure 5 has AfP privacy as in Definition 8.*

(See proof sketch in Section C).

An AfP based on the setup presented in Figure 4 will not provide a good foundation for YOSO-MPC or even just a proactive secret sharing scheme. The reason is, that as soon as a party ID_i publishes an AfP tag, any other party can verify that ID_i won the lottery and, thus, link the identity of ID_i to the public key $\mathsf{pk}_{\mathsf{Anon},\psi(i)}$ from the output of the mixnet. This will ruin the setup for this party when future lotteries are conducted. More importantly, a powerful adversary is able to identify any subsequent ECW ciphertexts towards this party and can design its corruption strategy accordingly. What we want is a new ephemeral public key $\mathsf{pk}_{\mathsf{Anon},\psi(i)}$ for each party and for each slot sl in the blockchain execution where an AfP is produced. Note that a new lottery setup is necessary for each slot sl even though different parties are producing AfP tags in different slots. The reason is that observing *any* AfP tag, inadvertently, skews the probability distribution and helps the adversary in guessing future lottery winner.
A simple way to solve the above issue is to repeat the lottery setup and obtain multiple vectors of the format $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$. Then, any party can use a new anonymized public key for each AfP tag. We describe this property as *bounded* AfP privacy. Bounded AfP privacy ensures that the AfP scheme can be executed a bounded number times before lottery winners can be linked to specific public keys in the setup and ECW ciphertexts starts betraying their receivers. Note that the idea of generating multiple lottery setups in batches (preprocessing) can result in more efficient protocols. But it has the downside that, while using the preprocessed public keys, the number of parties in the system is static.
In Section 6 we look at how to use the ECW and AfP in an anonymous PVSS protocol where we want encrypt towards multiple parties. In such a setting we can use linkable ring signatures (Section D) to proof membership in a committee without directly revealing our public key in the setup.

3.4 AfP with Reusable Setup

In Appendix D, we describe an efficient NIZK that allows for a party ID_i to prove knowledge of a lottery secret key $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_j, \mathsf{sk}_{L,i}) = 1$ for $\mathsf{P}_j \in \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ without revealing P_j. Using this NIZK and an anonymous channel, we can construct an AfP that can be used multiple times without linking a party P_i to its setup public key. In order to generate an AfP on message m on behalf of role P in slot sl, P_i with $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$ first generates a NIZK π proving knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_j, \mathsf{sk}_{L,i}) = 1$ for $\mathsf{P}_j \in \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. Now P_i generates an SoK σ on the message m of knowledge of a valid proof π for the aforementioned statement. ID_i publishes σ through an anonymous channel, avoiding its identity to be linked to the set $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. The security and privacy guarantees for this AfP follow in a straightforward way from our previous analysis. While using this construction has a clear extra cost in relation to our simple AfP, we show in Appendix D.2 how to efficiently perform such a reusable setup AfP on a set of ciphertexts, which is useful for our resharing application.

4 Publicly Verifiable Secret Sharing

4.1 Model

We define a publicly verifiable secret sharing (PVSS) scheme with t privacy and $t+1$-reconstruction, based on the models provided in [27,25,20,8]. The goal is for a dealer to share a secret $S \in \mathbb{G}$ to a set of n parties $\mathcal{P} = \{P_1, \cdots, P_n\}$, so that $t+1$ shares will be needed to reconstruct the secret and no information will be revealed from t shares. We require public verifiability for correctness of sharing by the dealer, and for reconstruction of the secret by a set of t+1 parties. Due to this requirement, the protocol is entirely carried out using a public ledger as means of communication, and we introduce an additional entity, the verifier, whose task is to check the correctness of the outputs of the dealer and parties.

We provide the syntax below. A modification we introduce with respect to the usual model is that we include asymmetric key pairs for dealers and an additional initial round where the parties can broadcast an ephemeral public key. This can allow for more efficient constructions such that the benefits are worth the extra round of communication, as we will see in Section 4.3.

- *Setup* The algorithm Setup, on input the security parameter 1^λ, outputs the public parameters pp. The dealer generates their key pair with $\mathsf{DKeyGen}$, which on input the parameters pp, outputs their own key pair $(\mathsf{pk}_D, \mathsf{sk}_D)$. Each party $P_i \in \mathcal{P}$ performs KeyGen, which on input the parameters pp, and a unique identifier id_i, outputs their own public/ private key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$, and shares their public key. The outputs of each party P_i can be verified with $\mathsf{VerifyKey}$, which, on input the public parameters pp, id_i, and the public key of the party pk_i, outputs 1 if pk_i is valid. Let the sets $\mathcal{PK}_D$ and $\mathcal{PK}$ contain all key pairs output by $\mathsf{DKeyGen}$ and KeyGen respectively.
- *Distribution* The dealer performs Dist which, on input pp, their key pair $(\mathsf{pk}_D, \mathsf{sk}_D)$, the public keys pk_i for each $P_i \in \mathcal{P}$, and the secret $S \in \mathbb{G}$, outputs an encrypted share C_i for each party P_i and a proof $\mathsf{Pf}_{\mathsf{Sh}}$ that these are encryptions of a valid sharing of a secret.
- *Verification* A verifier performs Verify which, on input pp, pk_D, $\{(\mathsf{pk}_i, C_i) : i \in [n]\}$ the public keys and encrypted shares of all parties P_i, and the proof $\mathsf{Pf}_{\mathsf{Sh}}$, outputs 1 if they are valid.
- *Reconstruction* This phase consists of two subphases.
  - *Decryption of the Shares* Let $\mathcal{Q}$ be the set of parties that decrypt their shares. Each party $P_i \in \mathcal{Q}$ performs $\mathsf{DecShare}$ which, on input pp, pk_D, their public and secret key $\mathsf{pk}_i, \mathsf{sk}_i$, and the encrypted share C_i, outputs a decrypted share A_i and a proof $\mathsf{Pf}_{\mathsf{Dec}i}$ of correct decryption.
  - *Pooling of the Shares* A verifier first checks there are enough valid decrypted shares to pool together. For each $i \in Q$, they perform $\mathsf{VerifyDec}$ which, on input pp, pk_D, a public key pk_i, an encrypted share C_i, a decrypted share A_i and proof $\mathsf{Pf}_{\mathsf{Dec}i}$, outputs 1 only if A_i is a valid decryption of C_i. Let the set $\mathcal{T}$ consist of all parties whose decrypted shares pass verification. The Verifier performs Rec which, on input pp and the decrypted shares A_i for $i \in \mathcal{T}$, outputs the reconstructed secret S.

For non–deterministic algorithms we sometimes explicitly reference the randomness r input. For example, $\mathsf{Dist}(pp, \mathsf{pk}_D, sk_D, \{\mathsf{pk}_i : i \in [n]\}, S; r)$.

We require a PVSS to satisfy correctness, verifiability and IND1-secrecy.

Definition 9 (Correctness). *A PVSS satisfies* correctness *if for each secret $S \in \mathbb{G}$ and for any set of identifiers $\{id_i : i \in [n]\}$*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, \mathsf{sk}_D) \leftarrow \mathsf{DKeyGen}(pp); \\ \forall i \in [n] \quad (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(pp, id_i); \\ (\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \leftarrow \\ \mathsf{Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S); \\ \forall i \in [n] \quad (A_i, \mathsf{Pf}_{\mathsf{Dec}i}) \leftarrow \\ \mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}_i, \mathsf{sk}_i, C_i); \\ S' \leftarrow \mathsf{Rec}(pp, \{A_i : i \in [n]\}); \end{array} : \begin{array}{l} \forall i \in [n] \mathsf{VerifyKey}(pp, id_i, \mathsf{pk}_i) = 1 \\ \wedge \mathsf{Verify}(pp, \mathsf{pk}_D, \\ \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1 \\ \wedge \forall i \in [n] \mathsf{VerifyDec}(pp, \mathsf{pk}_D, \\ \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i}) = 1 \\ \wedge S' = S \end{array}\right]$$

$$= 1.$$

Verifiability The verifiability requirement ensures that an adversary must honestly follow the protocol. This means that it can be verified that parties honestly generate their ephemeral public keys (key generation), the dealer outputs encrypted shares for a secret (distribution), and that the parties honestly decrypt their shares in reconstruction (decryption).

Definition 10 (Verifiability of Key Generation). *A PVSS satisfies* verifiability of key generation *if there exists a negligible function $\mu(\lambda)$such that*

$$\left|\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, id, \mathsf{pk}) \leftarrow \mathcal{A}(pp); \end{array} : \begin{array}{c} \mathsf{VerifyKey}(pp, id, \mathsf{pk}) = 1 \\ \wedge \exists \mathsf{sk}\ s.t.\ (\mathsf{sk}, \mathsf{pk}) \notin \mathcal{PK} \end{array}\right]\right| \leq \mu(\lambda).$$

Definition 11 (Verifiability of Distribution). *A PVSS satisfies* verifiability of distribution *if there exists a negligible function $\mu(\lambda)$such that*

$$\left|\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \\ \leftarrow \mathcal{A}(pp); \end{array} : \begin{array}{l} \mathsf{Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \\ = 1 \\ \wedge \exists S, \mathsf{sk}_D, r\ \ s.t. \\ ((\mathsf{sk}_D, \mathsf{pk}_D) \notin \mathcal{PK}_D \vee \\ \mathsf{Dist}(pp, \mathsf{pk}_D, sk_D, t, \{\mathsf{pk}_i : i \in [n]\}, S; r) \\ \neq (\{C_i : i \in [n]\}, \cdot)) \end{array}\right]\right|$$

$$\leq \mu(\lambda).$$

Definition 12 (Verifiability of Decryption). *A PVSS satisfies* verifiability of decryption *if there exists a negligible function $\mu(\lambda)$such that*

$$\left|\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, \mathsf{pk}, C, A, \mathsf{Pf}_{\mathsf{Dec}}) \leftarrow \mathcal{A}(pp); \end{array} : \begin{array}{l} \mathsf{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk}, C, A, \mathsf{Pf}_{\mathsf{Dec}}) = 1 \\ \wedge \exists (\mathsf{sk}, r)\ \ s.t. ((\mathsf{sk}, \mathsf{pk}) \notin \mathcal{PK} \vee \\ \mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C; r) \neq (A, \cdot)) \end{array}\right]\right|$$

$$\leq \mu(\lambda).$$

Algorithm 12 $\mathrm{Game}^{\mathsf{ind\text{-}secrecy},b}_{\mathcal{A},\mathsf{PVSS}}$

```
procedure DIST((U, S'))
    if U ⊄ [n+1, k] or |U| ≠ n then
        return ⊥
    end if
    ({C'_i : i ∈ [n]}, Pf_Sh) ← Dist(pp, pk_D, sk_D, t, n, {pk_i : i ∈ U}, S')
    return ({C'_i : i ∈ [n]}, Pf_Sh)
end procedure
procedure Game^{ind-secrecy,b}_{A,PVSS}(λ)
    pp ← Setup(1^λ, t, n), (pk_D, sk_D) ← DKeyGen(pp)
    ∀i ∈ [n − t]   (pk_i, sk_i) ← KeyGen(pp, i)
    ({pk_i : i ∈ [n − t + 1, n]}, {pk_i : i ∈ [n + 1, k]}) ← A(pp, pk_D, {pk_i : i ∈ [n − t]})
    if ∃i ∈ [n − t + 1, k] such that VerifyKey(pp, i, pk_i) = 0 then
        return 0
    end if
    S_0, S_1 ←$ G, ({C_i : i ∈ [n]}, Pf_Sh) ← Dist(pp, pk_D, sk_D, t, {pk_i : i ∈ [n]}, S_0)
    b' ← A^DIST(S_b, {C_i : i ∈ [n]}, Pf_Sh)
    return b'
end procedure
```

Indistinguishability of Secrets (IND-1 Secrecy) We now present the IND-1 Secrecy definition from [8]. We have modified this definition to fit the adjusted syntax because Dist can no longer be performed by the adversary (as it takes sk_D as input). We now provide a DIST oracle that will return the outputs of the Dist algorithm. To capture that the public keys of the parties should be ephemeral, we do not allow the public keys of parties that are used in the challenge to be input to this oracle. We therefore allow the adversary to output an extra $n-k$ keys.

Definition 13 (IND-1 Secrecy). *A PVSS satisfies* indistinguishability of secrets *if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$such that*

$$\left|\Pr\left[\mathrm{Game}^{\mathsf{ind\text{-}secrecy},0}_{\mathcal{A},\mathsf{PVSS}}(\lambda)=1\right]-\Pr\left[\mathrm{Game}^{\mathsf{ind\text{-}secrecy},1}_{\mathcal{A},\mathsf{PVSS}}(\lambda)=1\right]\right|\leq\mu(\lambda).$$

4.2 Construction from a $\mathbb{Z}_p$-LHE Schemes

We present in Figure 6 our construction for a PVSS scheme HEPVSS based on a $\mathbb{Z}_p$-LHE scheme with proof of correct decryption. This construction does not require the dealer to hold a key pair or parties to prove honest generation of keys and therefore we remove this from the syntax. Moreover, because the dealer does not have a key pair, here we do not require the public keys pk_i to be ephemeral.

Setup (Algorithms 13 and 14) The public parameters output by Setup specify a prime p, a $\mathbb{Z}_p$-LHE $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with $\mathfrak{P} = \mathbb{G}$ a group of order p, and evaluation points for the secret sharing scheme.

Distribution (Algorithm 15) The dealer distributes a secret $S \in \mathbb{G}$ by first computing shares according to $\mathsf{GShamir.Share}$ (Figure 3). Then the dealer encrypts each share A_i under pk_i and publishes the encryptions C_i. The dealer publishes a proof of knowledge of message and randomness involed in the PVSS.

Verification (Algorithm 16) Correctness of the encrypted shares outputted by $\mathsf{HEPVSS.Dist}$ is verified.

Algorithms for Public Verifiable Secret Sharing Scheme HEPVSS

Algorithm 13 $\mathsf{HEPVSS.Setup}(1^\lambda, t, n)$

$(\mathbb{G}, G, p, \mathcal{E}) \leftarrow\$\, \mathcal{G}(1^\lambda)$
Choose pairwise distinct
$\alpha_0, \alpha_1, \cdots \alpha_n \in \mathbb{Z}_p$
return
$pp =$
$(\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\}, \mathcal{E})$

Algorithm 14 $\mathsf{HEPVSS.KeyGen}(pp, id)$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow\$\, \mathcal{E}.\mathsf{Gen}(1^\lambda)$
return $(\mathsf{sk}, \mathsf{pk})$

Algorithm 15 $\mathsf{HEPVSS.Dist}(pp, \{\mathsf{pk}_i : i \in [n]\}, S)$

Parse pp as $(\mathbb{G}, G, p, n, \{\alpha_i : i \in [0, n]\}, \mathcal{E}) := (pp_{\mathsf{Sh}}, \mathcal{E})$
$(\{A_i : i \in [n]\}, m(X)) \leftarrow \mathsf{GShamir}(pp_{\mathsf{Sh}}, S)$
for $i \in [n]$ **do**
$\rho_i \leftarrow\$\, \mathfrak{R}$, $C_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(A_i, \rho_i)$
end for
$\mathcal{W} \leftarrow \mathbb{G} \times \mathbb{Z}_p[X]_{\leq t} \times \mathfrak{R}^n, \quad \mathcal{X} \leftarrow \{0\} \times \mathfrak{C}^n, \quad pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
$w \leftarrow (S, m(X), \rho_1, \ldots, \rho_n), \quad x \leftarrow (0, C_1, \ldots, C_n)$
Let f given by
$f(w) := (m(\alpha_0), \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_1}(S + m(\alpha_1) \cdot G; \rho_1), \ldots, \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_n}(S + m(\alpha_n) \cdot G; \rho_n))$
$\mathsf{Pf}_{\mathsf{Sh}} \leftarrow \mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp_\pi, x, f)$
return $(\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

Algorithm 16 $\mathsf{HEPVSS.Verify}(pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

Define $\mathcal{W}, \mathcal{X}, pp_\pi, x, f$ as in $\mathsf{HEPVSS.Dist}$
return $\mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, x, f, \mathsf{Pf}_{\mathsf{Sh}})$

Algorithm 17 $\mathsf{HEPVSS.DecShare}(pp, \mathsf{pk}, \mathsf{sk}, C)$

$A \leftarrow \mathsf{Dec}_{\mathsf{sk}}(C)$
$\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathcal{E}.\mathsf{ProveDec}(A, C, \mathsf{pk})$
return $(A, \mathsf{Pf}_{\mathsf{Dec}})$

Algorithm 18 $\mathsf{HEPVSS.VerifyDec}(pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}\,i})$

return $\mathcal{E}.\mathsf{VerifyDec}(A_i, C_i, \mathsf{pk}_i, \mathsf{Pf}_{\mathsf{Dec}\,i})$

Algorithm 19 $\mathsf{HEPVSS.Rec}(pp, \{A_i : i \in \mathcal{T}\}$

return $\mathsf{GShamir.Rec}(pp, \{A_i : i \in \mathcal{T}\})$

Fig. 6. Algorithms for HEPVSS

Reconstruction Every party P_i decrypts their share A_i from C_i and prove correctness of decryption with Algorithm 17. Let $\mathcal{Q}$ be the parties that decrypt their shares. Validity of the decrypted shares is verified with $\mathsf{HEPVSS.VerifyDec}$ (Algorithm 18). The secret can be reconstructed from a set $\mathcal{T}$ of valid decrypted shares (Algorithm 19).

Security We prove that HEPVSS satisfies correctness, indistinguishability of secrets and verifiability in Appendix E.2.

4.3 An Optimized Construction based on Diffie-Hellman Key Exchange and SCRAPE

We now give an optimized construction DHPVSS for a PVSS with a more efficient proof of sharing correctness based on the SCRAPE technique from [8] combined with an idea explained below. The PVSS scheme has IND1-secrecy under the DDH assumption.

Let $A_i = a_i \cdot G$ be (purportedly) group Shamir shares for a secret $S \in \mathbb{G}$. A SCRAPE check (Theorem 1) consists on the verification

$$\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot a_i = 0,$$

or alternatively

$$\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot A_i = O,$$

for O the identity element of $\mathbb{G}$. Here v_i are fixed coefficients dependent on the α_i and $m^*(X)$ is sampled uniformly at random from $\mathbb{Z}_p[X]_{\leq n-t-1}$. Passing the check guarantees with high probability that $a_i = m(\alpha_i)$ for all $i \in [n]$ for some polynomial $m(X) \in \mathbb{Z}_p[X]_{\leq t}$.

In [8], the "encryptions"[5] of the shares A_i are $C_i = a_i \cdot \mathsf{pk}_i$ (for pk_i as in the El Gamal construction above). Because these are in different bases the check above cannot be directly applied on the C_i's and thus, for proving correctness, the dealer carries out the check on the values $M_i = a_i H$ (where H is another generator of the group for which nobody knows the $\mathsf{DL}_G(H)$) and needs to use DLEQ proofs to show M_i hides the same exponent as C_i, for all i. This leads to a similar proof size linear in n as our generic construction instantiated with El Gamal because it needs posting the n group elements M_i and DLEQ proofs.

Instead, we will reduce this size to *constant* by setting the encryptions to be $C_i = A_i + \mathsf{sk}_D E_i$ where $(\mathsf{sk}_D, \mathsf{pk}_D)$ is a key-pair for the dealer (of the same form $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$) and E_i is an element of pk_i in $\mathbb{G}$. These can be seen as El Gamal encryptions where the randomness is sk_D for all encryptions, i.e. the first element of the El Gamal encryption is pk_D for all encryptions. Alternatively, they can be seen as El Gamal encryptions under the dealer's public key with randomness given by sk_i. The advantage of this is that now the SCRAPE check

$$\sum_{i=1}^{n} v_i m^*(\alpha_i) A_i = O$$

is equivalent to checking

$$\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot C_i = \mathsf{sk}_D \cdot \left(\sum_{i=1}^{n} v_i m^*(\alpha_i) E_i \right).$$

But this is actually *one single* DLEQ proof

$$\mathsf{DLEQ}(\mathsf{sk}_D; G, \mathsf{pk}_D, U, V)$$

[5] Note this is a weak form of deterministic encryption which does not satisfy IND-CPA security

with

$$U = \sum_{i=1}^{n} v_i m^*(\alpha_i) E_i,$$

$$V = \sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot C_i.$$

One detail is that now the prover is the one who needs to sample $m^*(X)$ but this can be done by means of a random oracle

$$\mathcal{H} : \{0,1\}^* \to \{f^* \in \mathbb{Z}_p[X] : \mathsf{deg}(f^*) \leq n - t - 1\}.$$

The algorithms can be found in Figure 7 and Figure 8.

Algorithms for PVSS scheme DHPVSS, Setup and Distribution

Algorithm 20 $\mathsf{DHPVSS.Setup}(1^\lambda, t, n)$

$(\mathbb{G}, G, p) \leftarrow\$ \mathcal{G}(1^\lambda)$
Choose pairwise distinct
$\alpha_0, \alpha_1, \cdots \alpha_n \in \mathbb{Z}_p$
$\forall i \in [n] \quad v_i \leftarrow \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}$
return
$pp = (\mathbb{G}, G, p, t, n, \alpha_0, \{(\alpha_i, v_i) : i \in [n]\})$

Algorithm 21 $\mathsf{DHPVSS.DKeyGen}(pp)$

$\mathsf{sk}_D \leftarrow\$ \mathbb{Z}_p^*, \mathsf{pk}_D \leftarrow \mathsf{sk}_D \cdot G$
return $(\mathsf{pk}_D, \mathsf{sk}_D)$

Algorithm 22 $\mathsf{DHPVSS.KeyGen}(pp, id)$

$\mathsf{sk} \leftarrow\$ \mathbb{Z}_p^*, E \leftarrow \mathsf{sk} \cdot G$
$\Omega \leftarrow \mathsf{DL}(\mathsf{sk}; G, E, id)$
$\mathsf{pk} \leftarrow (E, \Omega)$
return $(\mathsf{pk}, \mathsf{sk})$

Algorithm 23 $\mathsf{DHPVSS.VerifyKey}(pp, id, \mathsf{pk})$

parse pk as (E, Ω)
return accept iff Ω is valid w.r.t G, E, id

Algorithm 24 $\mathsf{DHPVSS.Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, t, \{\mathsf{pk}_i : i \in [n]\}, S)$

parse pk_i as (E_i, Ω_i), pp as $(\mathbb{G}, G, p, t, n, \alpha_0, \{(\alpha_i, v_i) : i \in [n]\})$
$pp_{\mathsf{Sh}} \leftarrow (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\})$
$(\{A_i\}_{i \in [n]}, m(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathsf{Sh}}, S)$
$\forall i \in [n], C_i \leftarrow \mathsf{sk}_D \cdot E_i + A_i$
$m^* \leftarrow \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
$V \leftarrow \sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot E_i$
$\mathsf{Pf}_{\mathsf{Sh}} \leftarrow \mathsf{DLEQ}(\mathsf{sk}_D; G, \mathsf{pk}_D, U, V)$
return $(\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

Fig. 7. Algorithms for PVSS scheme DHPVSS, Setup and Distribution

Security We prove that DHPVSS satisfies correctness, indistinguishability of secrets and verifiability in Appendix E.3.

Algorithms for PVSS scheme DHPVSS, Verification and Reconstruction

Algorithm 25 $\mathsf{DHPVSS.Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

parse pk_i as (E_i, Ω_i), pp as $(\mathbb{G}, G, p, t, n, \{(\alpha_i, v_i) : i \in [n]\})$
$f^* \leftarrow \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
$V \leftarrow \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot E_i$
return accept iff $\mathsf{Pf}_{\mathsf{Sh}}$ is valid w.r.t G, pk_D, U, V

Algorithm 26 $\mathsf{DHPVSS.DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C)$

parse pk as (E, Ω)
$A' \leftarrow C - \mathsf{sk} \cdot \mathsf{pk}_D$
$\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathsf{DLEQ}(\mathsf{sk}; G, E, \mathsf{pk}_D, C - A')$
return $(A', \mathsf{Pf}_{\mathsf{Dec}})$

Algorithm 27 $\mathsf{DHPVSS.VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}\,i})$

parse pk_i as (E_i, Ω_i)
return accept iff $\mathsf{Pf}_{\mathsf{Dec}\,i}$ is valid w.r.t $G, E_i, \mathsf{pk}_D, C_i - A_i$

Algorithm 28 $\mathsf{DHPVSS.Rec}(pp, \{A_i : i \in \mathcal{T}\})$

return $\mathsf{GShamir.Rec}(pp, \{A_i : i \in \mathcal{T}\})$

Fig. 8. Algorithms for PVSS scheme DHPVSS, Verification and Reconstruction

Communication Complexity Comparison. The communication complexity of DHPVSS.Dist is $(n+2)\log p$ bits. In contrast, HEPVSS.Dist instantiated with El Gamal is of $(3n+3)\log p$ bits. The communication of both DHPVSS.DecShare and HEPVSS.DecShare is $3\log p$ bits. In contrast secret distribution in SCRAPE [8] requires $(3n+1)\log p$ bits. This was reduced to $(n+t+2)\log p$ bits in ALBATROSS [9]. The share decryption complexities in those papers are similar to ours. See Appendix F for details.

5 PVSS Resharing

In this section we introduce protocols that allow a committee $\mathcal{C}_{\mathsf{r}}$ of size n_{r}, among which a secret has been PVSSed with an underlying t_{r}-threshold Shamir scheme, to create a PVSS of the same secret for the next committee $\mathcal{C}_{\mathsf{r}+1}$ of size $n_{\mathsf{r}+1}$ and with threshold $t_{\mathsf{r}+1}$. In fact the committees do not need to have the same size, nor we need the same thresholds. By design, the protocols will keep the secret hidden from any adversary corrupting t_{r} parties of $\mathcal{C}_{\mathsf{r}}$ and $t_{\mathsf{r}+1}$ or less from $\mathcal{C}_{\mathsf{r}+1}$, and will be correct as long as there are $t_{\mathsf{r}}+1$ honest parties in $\mathcal{C}_{\mathsf{r}}$. In particular, this can be used a party P to transmit a message to a committee in the future, by keeping this secret being reshared among successive committees and setting the last Shamir threshold to be 0. Suppose for now that the secret sharing scheme is for secrets over $\mathbb{Z}_p$. In order to create a PVSS of the same secret towards the next committee $\mathcal{C}_{\mathsf{r}+1}$, we need a subcommittee L of $t_{\mathsf{r}}+1$ parties in $\mathcal{C}_{\mathsf{r}}$ to successfully reshare their shares among the next committee. Call the resharing of each $m(\alpha_\ell)$ (the vector containing each of the shares obtained by parties in the next committee) by $[m(\alpha_\ell)]_{\mathcal{C}_{\mathsf{r}+1}}$. By linearity of Shamir sharing it is then clear that $\sum_{\ell \in L} \lambda_{\ell,L}[m(\alpha_\ell)] = [\sum_{\ell \in L} \lambda_{\ell,L} m(\alpha_\ell)] = m(\alpha_0)$ where $\lambda_{\ell,L} := \prod_{j \in L, j \neq \ell} \frac{\alpha_0 - \alpha_j}{\alpha_\ell - \alpha_j}$.

In our situation, we need to do some adjustments. First, since we are considering Shamir sharing over groups of order p, the resharing party does not need to know the sharing polynomial necessarily, but rather samples $m(X)$ of degree at most t with $m(\beta_0) = 0$, and then lets $A_i = m(\beta_i) \cdot G + S$. Moreover, we need to define L as a subset of $t+1$ parties that have proved to reshare correctly. This now means, not only that the encrypted shares created by a party P_ℓ in L are consistent with each other, but also that they have as secret S the share that this party had received in the previous round, for which the rest of the parties see a ciphertext C.

5.1 Resharing for HEPVSS

In the case of HEPVSS, the techniques used are very similar to PVSSing a single secret by a given party. The difference is we in addition need parties to prove that the value they are resharing is the correct secret (encrypted by the previous committee), but this can be integrated easily if the encryption scheme has $\mathbb{Z}_p$-linear decryption.

Let $\mathsf{pk}_{[n]}$ denote the set $\{\mathsf{pk}_i : i \in [n]\}$. Similarly $C_{[n]}$ denote a set of ciphertexts $\{C_i : i \in [n]\}$ and $\rho_{[n]}$ denote a set of elements from the randomness space $\{\rho_i : i \in [n]\}$. Recall $D_C(\mathsf{sk}) := \mathsf{Dec}_{\mathsf{sk}}(C)$. Define the relation

$$\begin{aligned} R_{\mathsf{Reshare}} =& \{(m(X), \mathsf{sk}, \rho_{[n]}); (\mathsf{pk}, \mathsf{pk}_{[n]}, C, C_{[n]}) : \\ & F(\mathsf{sk}) = \mathsf{pk}, m(\beta_0) = 0, \mathsf{Enc}_{\mathsf{pk}_i}(m(\beta_i) \cdot G + D_C(\mathsf{sk}); \rho_i) = C_i \text{ for } i \in [n]\} \end{aligned}$$

We therefore define the resharing proof in Figure 9. The protocol for PVSS resharing is then constructed as in Figure 10.

Proof system HEPVSS.Reshare for correct resharing of encrypted secret

Algorithm 29 $\mathsf{HEPVSS.Reshare.Prove}((m(X), \mathsf{sk}, \rho_{[n]}); (pp, \mathsf{pk}, \mathsf{pk}_{[n]}, C, C_{[n]}))$

parse $pp = (\mathbb{G}, G, p, t, n, \{\beta_i : i \in [n]\})$
$\mathcal{W} \leftarrow \mathbb{Z}_p[X]_{\leq t} \times \mathcal{SK} \times \mathfrak{R}^n, \mathcal{X} \leftarrow \mathcal{PK} \times \mathfrak{C}^n$,
$pp' \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$, $w \leftarrow (m(X), \mathsf{sk}, \rho_1, \ldots, \rho_n)$, $x \leftarrow (0, \mathsf{pk}, C_1, \ldots, C_n)$,
Set f_C given by $f_C(w) := (m(\beta_0), F(\mathsf{sk}), \mathsf{Enc}_{pk_1}(A_1; \rho_1), \ldots, \mathsf{Enc}_{pk_1}(A_n; \rho_n))$
  where $A_i = m(\beta_i) \cdot G + D_C(\mathsf{sk})$
return $\pi \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp', x, f_C)$

Algorithm 30 $\mathsf{HEPVSS.Reshare.Verify}(pp, \mathsf{pk}, \mathsf{pk}_{[n]}, C, C_{[n]}, \pi)$

Set $\mathcal{W}, \mathcal{X}, pp', x, f_C$, as in Reshare.Prove
return $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp', x, f_C, \pi)$

Fig. 9. Proof HEPVSS.Reshare of correct resharing of encrypted secret

5.2 Resharing for DHPVSS

In the case of DHPVSS, the situation is slightly more complicated due to the fact that the encryption of shares involves the public key of the dealer, and that we want to integrate the SCRAPE trick to save communication.

Protocol for HEPVSS resharing

Participants: Disjoint committees $\mathcal{C}_{\mathsf{r}} = \{P_{\mathsf{r},1}, \ldots, P_{\mathsf{r},n_{\mathsf{r}}}\}$ and $\mathcal{C}_{\mathsf{r}+1} = \{P_{\mathsf{r}+1,1}, \ldots, P_{\mathsf{r}+1,n_{\mathsf{r}+1}}\}$.

Public information: A group $\mathbb{G}$ of prime order p, with generator G. A homomorphic encryption scheme $\mathcal{E}$ with $\mathbb{Z}_p$-linear decryption, with plaintext space $\mathbb{G}$. Public keys $\mathsf{pk}_{j,i}$ for that encryption scheme corresponding to parties $\mathsf{P}_{j,i}$ above ($j = \mathsf{r}, \mathsf{r}+1, 1 \leq i \leq n_{\mathsf{r}}$), where $\mathsf{P}_{j,i}$ knows the corresponding secret key $\mathsf{sk}_{j,i}$; thresholds t_{r}, $t_{\mathsf{r}+1}$. Evaluation points $(\alpha_0, \alpha_1, \ldots, \alpha_{n_{\mathsf{r}}})$, $(\beta_0, \beta_1, \ldots, \beta_{n_{\mathsf{r}+1}})$.
Input: Public ciphertexts $C_{\mathsf{r},i}$, where it is guaranteed that $C_{\mathsf{r},i} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{r},i}}(A_{\mathsf{r},i})$ such that $A_{\mathsf{r},i} = f_{\mathsf{r}}(\alpha_i) \cdot G$ for some polynomial f_{r} of degree $\leq t_{\mathsf{r}}$.

Output: A public output $(C_{\mathsf{r}+1,1}, \ldots, C_{\mathsf{r}+1,n_{\mathsf{r}+1}})$ and a proof π that, for all $k = 1, \ldots, n_{\mathsf{r}+1}$, $C_{\mathsf{r}+1,k} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{r}+1,i}}(A_{\mathsf{r}+1,k})$ such that $A_{\mathsf{r}+1,k} = f_{\mathsf{r}+1}(\beta_k) \cdot G$ for some polynomial $f_{\mathsf{r}+1}$ of degree $\leq t_{\mathsf{r}+1}$ and $f_{\mathsf{r}+1}(\beta_0) = f_{\mathsf{r}}(\alpha_0)$.

Protocol:

1. Let $pp_{\mathsf{r}+1} = (\mathbb{G}, G, p, t_{\mathsf{r}+1}, n_{\mathsf{r}+1}, \{\beta_i : i \in [0, n]\})$.
2. Resharing: For $i = 1, ..., n_{\mathsf{r}}$, $P_{\mathsf{r},i}$ does the following
   (a) Compute $A_{\mathsf{r},i} = \mathsf{Dec}_{\mathsf{sk}_{\mathsf{r},i}}(C_{\mathsf{r},i})$.
   (b) For $i = 1, ..., n_{\mathsf{r}}$, $P_{\mathsf{r},i}$ computes
   $(\{A_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, m(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathsf{r}+1}, A_{\mathsf{r},i})$
   (c) Sample $\rho_{i\to j} \in \mathfrak{R}$ for $j \in [n_{\mathsf{r}+1}]$ and let $\rho_{i\to[n_{\mathsf{r}+1}]} = \{\rho_{i\to j} : j \in [n_{\mathsf{r}+1}]\}$
   (d) Compute $C_{i\to j} = \mathsf{Enc}_{\mathsf{pk}_j}(A_{i\to j}; \rho_{i\to j})$ for $j \in [n_{\mathsf{r}+1}]$.
   (e) Compute
   $$\pi_i \leftarrow \mathsf{HEPVSS.Reshare.Prove}\ (m(X), \mathsf{sk}, \rho_{i\to[n_{\mathsf{r}+1}]}; pp, \mathsf{pk}_{\mathsf{r},i}, \{\mathsf{pk}_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, C_{\mathsf{r},i}, \{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\})$$
   (f) Output $\{C_{i\to j} : j \in \mathcal{C}_{\mathsf{r}+1}\}, \pi_i$
3. Reconstruction of next share encryptions: each party in $\mathcal{P}$ locally constructs the encryptions of the shares for the following round as follows:
   (a) Define L containing the first $t+1$ indices i for which the following accepts:
   $$\mathsf{HEPVSS.Reshare.Verify}(\ pp, \mathsf{pk}_{\mathsf{r},i}, \{\mathsf{pk}_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, C_{\mathsf{r},i}, \{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, \pi_i)$$
   (b) For $j \in [n_{\mathsf{r}+1}]$, set $C_{\mathsf{r}+1,j} = \sum_{\ell\in L} \lambda_{\ell,L} C_{\ell\to j}$ [a]
   (c) Output $\{(C_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, (\pi_{\mathsf{r},\ell})_{\ell\in L})$.

[a] Here $\sum$ refers to the summatory with respect to the homomorphic operation on ciphertexts $\boxplus_{\mathfrak{C}}$

Fig. 10. Protocol for HEPVSS PVSS resharing

We proceed as follows: At a given round r, we will make sure that the output encryptions are

$$C_{\mathsf{r}+1,i} = \mathsf{sk}_{\mathsf{r}+1,i} \cdot \mathsf{pk}_{D,L_{\mathsf{r}}} + \sum_{\ell\in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}} A_{\ell\to i}$$

where $\mathsf{pk}_{D,L_{\mathsf{r}}} = \sum_{\ell\in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}} \cdot \mathsf{pk}_{D_\ell}$, and in turn pk_{D_ℓ} is the public key used as dealer by party P_ℓ of committee $\mathcal{C}_{\mathsf{r}}$ and $A_{\ell\to i}$ is the i-th share of the resharing of party $P_{\mathsf{r},\ell}$. Note $\mathsf{pk}_{D,L_{\mathsf{r}}}$ is computable publicly by any party. The element $A_i = \sum_{\ell\in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}} A_{\ell\to i}$ is the share of party $P_{\mathsf{r}+1,i}$ which this party can recover.

At round r, we have the values for the previous round $C_{\mathsf{r},j}$ and the challenge is to go from $\{C_{\mathsf{r},j} : j \in [n_{\mathsf{r}}]\}$ (created by parties in $L_{\mathsf{r}-1}$) to $\{C_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}$ (created by the parties in L_{r}).

A party in committee $\mathcal{C}_\mathsf{r}$ knows therefore some value $S = C - \mathsf{sk} \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$ where sk is the secret key for decrypting shares, and needs to create shares A_i of S and encrypt them using the public keys $\mathsf{pk}_{[n_{\mathsf{r}+1}]} = \{\mathsf{pk}_j : j \in [n_{\mathsf{r}+1}]\}$ of the parties of the next round and its own secret key sk_D (i.e. this party will create $C_{[n_{\mathsf{r}+1}]} = \{C_j : j \in [n_{\mathsf{r}+1}]\}$ with $C_j = \mathsf{sk}_D \cdot \mathsf{pk}_j + A_j$) and prove their validity. In conclusion we need a proof for the following relation

$$\begin{aligned} R_{\mathsf{DHPVSS,Reshare}} = \{&(m(X), \mathsf{sk}, \mathsf{sk}_D); (pp, \mathsf{pk}, \mathsf{pk}_D, \mathsf{pk}_{D,L}, \mathsf{pk}_{[n]}, C, C_{[n]}) : \\ &\mathsf{pk} = \mathsf{sk} \cdot G,\ \mathsf{pk}_D = \mathsf{sk}_D \cdot G,\ m(X) \in \mathbb{Z}_p[X]_{\leq t},\ m(\beta_0) = 0, \\ &\forall j \in [n], C_j = \mathsf{sk}_D \cdot \mathsf{pk}_j + A_j, \\ &\text{ where } A_j = (C - \mathsf{sk} \cdot \mathsf{pk}_{D,L}) + m(\beta_j) \cdot G\} \end{aligned}$$

However, we want to use the SCRAPE technique to reduce the size of the witness and hence of the proof. Note that if we set $U_j = C_j - \mathsf{sk}_D \cdot \mathsf{pk}_j - C + \mathsf{sk} \cdot \mathsf{pk}_{D,L}$, $j \in [n]$ and $U_0 = O$, we want to make sure that $U_j = m(\beta_j) \cdot G$ for a polynomial of degree $\leq t$ (in addition to the conditions $\mathsf{pk} = \mathsf{sk} \cdot G, \mathsf{pk}_D = \mathsf{sk}_D \cdot G$).

For $j \in [0, n]$, let

$$v'_j = \prod_{k \in [0,n] \setminus \{j\}} (\beta_j - \beta_k)^{-1}.$$

Observe these are not exactly the same coefficients as in the description of DHPVSS in Section 4.3 because they include the evaluation point β_0. By Theorem 1, we want to prove $\sum_{j=0}^{n} v'_i m^*(\beta_j) U_j = O$, for a random polynomial m^* of degree $n - t$.

Observe $\sum_{j=0}^{n} v'_j m^*(\beta_j) U_j = U' - \mathsf{sk}_D \cdot V' + W' \cdot (\mathsf{sk} \cdot \mathsf{pk}_{D,L} - C)$ with

$$U' := \sum_{j=1}^{n} v'_j m^*(\beta_j) C_j,\ V' := \sum_{j=1}^{n} v'_j m^*(\beta_j) \mathsf{pk}_j,\ W' := \sum_{j=1}^{n} v'_j m^*(\beta_j),$$

This reduces the problem to a proof of knowledge for

$$\begin{aligned} R'_{\mathsf{DHPVSS,Reshare},m^*} = \{&(\mathsf{sk}, \mathsf{sk}_D); (\mathsf{pk}, \mathsf{pk}_D, U') : \\ &\mathsf{pk} = \mathsf{sk} \cdot G, \mathsf{pk}_D = \mathsf{sk}_D \cdot G, U' - W' \cdot C = \mathsf{sk}_D V' - \mathsf{sk} W' \mathsf{pk}_{D,L}\} \end{aligned}$$

This leads to the protocol for PVSS resharing in Figure 11

6 Anonymous PVSS via ECW and AfP

In this section, we show how to construct PVSS (and re-sharing) for anonymous committees by instantiating our previous PVSS constructions using our ECW and AfP schemes. We start by showing how our previous protocols can be adapted to work with ECW and AfP instead of standard encryption and authentication. We then show how the optimizations in the DDH based constructions via the SCRAPE trick carry over to our anonymous setting if we instantiate our ECW and AfP schemes from similar assumptions. The protocols we construct in this section work in the YOSO model supporting up to $t < n/2$ corrupted parties and can be used as efficient building blocks for the protocols of [2,16].

Protocol for DHPVSS resharing

Participants: Disjoint committees $\mathcal{C}_{\mathsf{r}} = \{P_{\mathsf{r},1}, \ldots, P_{\mathsf{r},n_{\mathsf{r}}}\}$ and $\mathcal{C}_{\mathsf{r}+1} = \{P_{\mathsf{r}+1,1}, \ldots, P_{\mathsf{r}+1,n_{\mathsf{r}+1}}\}$.

Public information: A group $\mathbb{G}$ of prime order p, with generator G. Key pairs $(\mathsf{sk}_{D_{j,i}}, \mathsf{pk}_{D_{j,i}} = \mathsf{sk}_{D_{r,i}} \cdot G)$, $r = \mathsf{r}-1, \mathsf{r}$, $i \in [n_j]$ and key-pairs $(\mathsf{sk}_{r,i}, \mathsf{pk}_{r,i} = \mathsf{sk}_{r,i} \cdot G)$ for that encryption scheme corresponding to parties $\mathsf{P}_{r,i}$ above ($r = \mathsf{r}, \mathsf{r}+1, 1 \leq i \leq n_{\mathsf{r}}$), where $\mathsf{P}_{r,i}$ knows the corresponding secret key $\mathsf{sk}_{r,i}$; thresholds t_{r}, $t_{\mathsf{r}+1}$. Evaluation points $(\alpha_0, \alpha_1, \ldots, \alpha_{n_{\mathsf{r}}})$, $(\beta_0, \beta_1, \ldots, \beta_{n_{\mathsf{r}+1}})$. Random oracles $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p[X]_{\leq n-t}$, $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$.
Input: Public ciphertexts $C_{\mathsf{r},i}$, where it is guaranteed that $C_{\mathsf{r},i} = \mathsf{sk}_{\mathsf{r},i}\mathsf{pk}_{D,L_{\mathsf{r}-1}} + A_{\mathsf{r},i}$ such that $A_{\mathsf{r},i} = h_{\mathsf{r}}(\alpha_i) \cdot G$ for some polynomial h_{r} of degree $\leq t_{\mathsf{r}}$ and $\mathsf{pk}_{D,L_{\mathsf{r}-1}} = \sum_{\ell \in L_{\mathsf{r}-1}} \lambda_{\ell,L_{\mathsf{r}-1}} \mathsf{pk}_{D_{\mathsf{r}-1,\ell}}$.

Output: A public output $(C_{\mathsf{r}+1,1}, \ldots, C_{\mathsf{r}+1,n_{\mathsf{r}+1}})$ and a proof π that, for all $j = 1, \ldots, n_{\mathsf{r}+1}$, $C_{\mathsf{r}+1,j} = \mathsf{sk}_{\mathsf{r}+1,j}\mathsf{pk}_{D,L_{\mathsf{r}}} + A_{\mathsf{r}+1,j}$ such that $A_{\mathsf{r}+1,j} = h_{\mathsf{r}+1}(\beta_j) \cdot G$ for some polynomial $h_{\mathsf{r}+1}$ of degree $\leq t_{\mathsf{r}+1}$ and $h_{\mathsf{r}+1}(\beta_0) = h_{\mathsf{r}}(\alpha_0)$.

Protocol:

1. Let $pp_{\mathsf{r}+1} = (\mathbb{G}, G, p, t_{\mathsf{r}+1}, n_{\mathsf{r}+1}, \{\beta_i : i \in [0,n]\})$
2. Resharing: For $i = 1, ..., n_{\mathsf{r}}$, $P_{\mathsf{r},i}$ does the following
   (a) $\mathsf{pk}_{D,L_{\mathsf{r}-1}} \leftarrow \sum_{\ell \in L_{\mathsf{r}-1}} \lambda_{\ell,L_{\mathsf{r}-1}} \mathsf{pk}_{D_{\mathsf{r}-1,\ell}}$
   (b) $A_{\mathsf{r},i} \leftarrow C_{\mathsf{r},i} - \mathsf{sk}_{\mathsf{r},i}\mathsf{pk}_{D,L_{\mathsf{r}-1}}$.
   (c) $(\{A_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, m(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathsf{r}+1}, A_{\mathsf{r},i})$
   (d) $C_{i\to j} \leftarrow \mathsf{sk}_{D_{\mathsf{r}},i}\mathsf{pk}_{\mathsf{r}+1,j} + A_{i\to j}$ for $j \in [n_{\mathsf{r}+1}]$.
   (e) $m^*(X) \leftarrow \mathcal{H}(\{C_{\mathsf{r},i} : i \in [n_{\mathsf{r}}]\}, \mathsf{pk}_{D,L_{\mathsf{r}-1}})$
   (f) $U'_i \leftarrow (\sum_{j=1}^{n} v'_j m^*(\beta_j) C_{i\to j})$,
   (g) $V'_i \leftarrow \sum_{j=1}^{n} v'_j m^*(\beta_j)\mathsf{pk}_j$
   (h) $W'_i \leftarrow \sum_{j=1}^{n} v'_j m^*(\beta_j)$
   (i) $\mathcal{W} \leftarrow \mathbb{Z}_p^2$, $\mathcal{X} \leftarrow \mathbb{G}^3$, $pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H}')$
   (j) Compute $\pi_{\mathsf{r},i} \leftarrow \mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Prove}((\mathsf{sk}, \mathsf{sk}_D); pp_\pi, (\mathsf{pk}, \mathsf{pk}_D, U'_i - W'_i C_{\mathsf{r},i}), f_i)$
   where $f_i(\mathsf{sk}, \mathsf{sk}_D) := (\mathsf{sk} \cdot G, \mathsf{sk}_D \cdot G, \mathsf{sk}_D V'_i - \mathsf{sk} W'_i \mathsf{pk}_{D,L_{\mathsf{r}-1}})$
   (k) Output $\{C_{i\to j} : j \in \mathcal{C}_{\mathsf{r}+1}\}, \pi_{\mathsf{r},i}$
3. Reconstruction of next share encryptions: each party in $\mathcal{P}$ locally constructs the encryptions of the shares for the following round as follows:
   (a) For each $i \in \mathcal{C}_{\mathsf{r}}$:
      i. Compute U'_i, W'_i and f_i from the output $\{C_{i\to j} : j \in \mathcal{C}_{\mathsf{r}+1}\}$ of $P_{\mathsf{r},i}$ and the public information $\mathcal{H}, \mathsf{pk}_{D,L_{\mathsf{r}-1}}, C_{\mathsf{r},i}$, $\{\mathsf{pk}_j, \beta_j : j \in \mathcal{C}_{\mathsf{r}+1}\}$ as $P_{\mathsf{r},i}$ does in the resharing protocol above. Set pp_π as above.
      ii. Compute $\mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi(\mathsf{pk}, \mathsf{pk}_D, U'_i - W'_i C_{\mathsf{r},i})), f_i, \pi_{\mathsf{r},i})$
   (b) Define L_{r} the set of $t+1$ first indices for which the above proofs accept.
   (c) For $j \in [n_{\mathsf{r}+1}]$, set $C_{\mathsf{r}+1,j} = \sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L} C_{\ell\to j}$
   (d) Output $\{(C_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, (\pi_{\mathsf{r},\ell})_{\ell \in L_{\mathsf{r}}})$.

Fig. 11. Protocol for DHPVSS resharing

In the previous sections, we have constructed both a PVSS scheme (Section 4.2) and a PVSS re-sharing scheme (Section 5.1) based on $\mathbb{Z}_p$-linear encryption schemes (as defined in Section 2.2). Despite being efficient, these constructions are not fit for the YOSO model because they require the dealer to know the public keys of the parties who will receive shares, consequently revealing their identities. In order to solve this issue, we show that these protocols can also be instantiated with the ECW scheme of Section 3 even though they were designed to be instantiated with a $\mathbb{Z}_p$-linear encryption scheme. The core idea is that our ECW preserves all the properties of the underlying $\mathbb{Z}_p$-linear encryption scheme while adding the ability to encrypt towards a role rather than towards a party who owns a public key.

6.1 Constructing HEPVSS with ECW

We modify HEPVSS to use our ECW scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ for lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$ from Section 3 instead of a $\mathbb{Z}_p$-linear encryption scheme. Departing from the HEPVSS algorithms described in Figure 6, we make the following modifications:

- **Communication:** All messages are posted to the underlying blockchain ledger used by the ECW scheme $\mathcal{E}$.
- $\mathsf{HEPVSS.Setup}(1^\lambda)$: Besides the original setup parameters, we assume that n distinct role identifiers $\mathsf{P}_1, \ldots, \mathsf{P}_n$ are available and that an underlying blockchain protocol Γ is executed.
- $\mathsf{HEPVSS.KeyGen}(pp, id)$: Instead of publishing pk_i, each party P_i provides pk_i as input to the mixnet assumed as setup for $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$ and associated ECW scheme $\mathcal{E}$. The mixnet output $\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}$ is assumed to be available on the underlying blockchain and accessible as
$$(\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}, \eta) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl}).$$
Party P_i sets $\mathsf{sk}_{L,i} \leftarrow (\mathsf{pk}_{\mathcal{E},i}, \mathsf{sk}_{\mathcal{E},i})$.
- $\mathsf{HEPVSS.Dist}(pp, \{\mathsf{pk}_i : i \in [n]\}, S)$: Instead of computing $C_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(A_i, \rho_i)$, the dealer computes $C_i \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_i, A_i)$ using randomness ρ_i . Notice that this is equivalent to computing $C_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{Anon},j}}(A_i, \rho_i)$ for a j such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_i, \mathsf{sk}_{L,j}) = 1$. Hence, $\mathsf{Pf}_{\mathsf{Sh}}$ can still be computed via the same procedure. The dealer publishes
$$(\{C_i : i \in [n]\}, \{\mathsf{pk}_{\mathsf{Anon},j} : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}).$$
Notice that the public key $\mathsf{pk}_{\mathsf{Anon},j}$ used to generate each C_i is publicly known due to the structure of the lottery scheme.
- $\mathsf{HEPVSS.Verify}(pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$: No modification is needed, since $(\{C_i : i \in [n]\}, \{\mathsf{pk}_{\mathsf{Anon},j} : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$ has the same structure as in the original protocol.
- $\mathsf{HEPVSS.DecShare}(pp, \mathsf{pk}_j, \mathsf{sk}_{L,j}, C_i)$: Party P_j checks that its lottery witness $\mathsf{sk}_{L,j}$ is such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_i, \mathsf{sk}_{L,j}) = 1$ and, if yes, computes $A_i \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, C_i, \mathsf{sk}_{L,j})$. Proof $\mathsf{Pf}_{\mathsf{Dec}}$ is generated as in the original protocol. Notice that this procedure is also equivalent to generating an AfP $\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathsf{AfP.Sign}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}_i, \mathsf{sk}_{L,j}, A_i)$.
- $\mathsf{HEPVSS.VerifyDec}(pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}i})$: Proof $\mathsf{Pf}_{\mathsf{Dec}}$ is checked as in the original protocol. Notice that this procedure is also equivalent to generating an AfP $\{0, 1\} \leftarrow \mathsf{AfP.Ver}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}_i, \mathsf{Pf}_{\mathsf{Dec}}, A_i)$.
- $\mathsf{HEPVSS.Rec}(pp, \{A_i : i \in \mathcal{T}\}$: No modification is needed.

Due to the properties of the ECW scheme and the underlying lottery scheme, shares are encrypted towards parties randomly chosen to perform each role P_i whose identity remains unknown during the share distribution and verification phases. In case a reconstruction happens, parties executing each role reveal themselves by proving correctness of decrypted shares, which constitutes an AfP since it involved proving knowledge of $\mathsf{sk}_{L,j}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_i, \mathsf{sk}_{L,j}) = 1$.

6.2 Constructing Resharing for HEPVSS with ECW

In the context of resharing, the parties selected to execute roles $\mathsf{P}_1, \ldots, \mathsf{P}_n$ in slot sl_r wish to publicly verifiable reshare the secret whose shares they received towards roles $\mathsf{P}'_1, \ldots, \mathsf{P}'_{n'}$ in a future slot $\mathsf{sl}_{\mathsf{r}+1}$. In practice, this means that the resharing information will be received by a new randomly selected set of anonymous parties performing these roles in the future. Once again we explore the fact that our ECW inherits the properties of the underlying $\mathbb{Z}_p$-linear encryption scheme to modify the resharing protocol of Figure 10 to work with ECW.

We show how to modify the description of Figure 10 to obtain an ECW based resharing protocol:

- **Participants:** Parties executing roles $\mathsf{P}_1, \ldots, \mathsf{P}_n$ in slot sl_r and parties executing roles $\mathsf{P}'_1, \ldots, \mathsf{P}'_{n'}$ in slot $\mathsf{sl}_{\mathsf{r}+1}$.
- **Input:** Public (*i.e.* published in the underlying blockchain) ECW ciphertexts $C_i \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}_\mathsf{r}, \mathsf{P}_i, A_{tr,i})$ such that $A_{\mathsf{r},i} = f_\mathsf{r}(\alpha_i) \cdot G$ for some polynomial f_r of degree $\leq t_\mathsf{r}$.
- **Output:** ECW ciphertexts $C_i \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}_{\mathsf{r}+1}, \mathsf{P}'_i, A_{\mathsf{r},i})$ published in the underlying blockchain such that $A_{\mathsf{r}+1,k} = f_{\mathsf{r}+1}(\beta_k) \cdot G$ for some polynomial $f_{\mathsf{r}+1}$ of degree $\leq t_{\mathsf{r}+1}$ and $f_{\mathsf{r}+1}(\beta_0) = f_\mathsf{r}(\alpha_0)$.
- **Protocol:**
  - *Encryption/Decryption:* When decrypting ciphertexts using key sk_i for $i \in [n_\mathsf{r}]$, ECW decrypt using $\mathsf{sk}_{L,j}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}_\mathsf{r}, \mathsf{P}_i, \mathsf{sk}_{L,j}) = 1$. When encrypting a message under public key pk_j for $j \in [n_{\mathsf{r}+1}]$, ECW encrypt towards role P'_j in slot $\mathsf{sl}_{\mathsf{r}+1}$ using randomness $\rho_{\mathsf{r}+1,,j}$: $C_j \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}_{\mathsf{r}+1}, \mathsf{P}_{\mathsf{r}+1,j}, A)$. Notice that this is equivalent to computing $C_j \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{Anon},\mathsf{r}+1,j}}(A, \rho_{\mathsf{r}+1,j})$ for a j such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}_{\mathsf{r}+1}, \mathsf{P}_j, \mathsf{sk}_{L,j}) = 1$.
  - *Proof* $\mathsf{HEPVSS.Reshare.Verify}(pp, \mathsf{pk}_{\mathsf{r},i}, \{\mathsf{pk}_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, C_{\mathsf{r},i}, \{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, \pi_i)$*:* Notice that the structure of the ECW ciphertexts is compatible with this proof, so that it can be generated as in the original protocol. Analogously, this proof can also be verified as in the original protocol. Moreover, notice that it also acts as an AfP for ciphertexts $\{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}$ on behalf of role P_i of slot sl_r, since it requires knowledge of a $\mathsf{sk}_{L,j}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}_i, \mathsf{sk}_{L,j}) = 1$.

As in the PVSS with ECW protocol, due to the properties of the ECW scheme and the underlying lottery scheme, resharing information is encrypted towards parties randomly chosen to perform each role $\mathsf{P}_{\mathsf{r}+1,j}$ whose identity remains unknown until they act (*e.g.* by reconstructing the secret).

6.3 Efficient DDH-based Instantiation via DHPVSS

The most efficient instantiations of our techniques are obtained when using a variant of the El Gamal encryption scheme together with the SCRAPE share validity check. In order to enjoy the efficiency improvement, we show that our ECW is also compatible with these optimizations .

- **Setup and Lottery Predicate :** We use the same setup, *i.e.* we assume the parties have access to an ideal mixnet and input their public keys E_i so that the output of a tuple $\{(j, E_{\mathsf{Anon},j}) : j \in [n]\}$ which is written on the blockchain and accessible to every party through param function. The lottery predicate works the same way, having parties check whether $E_{\mathsf{Anon},k} = E_i$ for $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{P}||\eta)$ in order to determine if they have been selected for role P in slot sl. Moreover, every party publishes on the underlying blockchain a public key $\mathsf{pk}_{D,i}$ for which they know the corresponding secret key $\mathsf{sk}_{D,i}$, which they will use when encrypting.
- **Encryption:** As in our original ECW a party P_i encrypting m towards role P in slot sl starts by running $\mathsf{param}(\mathbf{B}, \mathsf{sl})$ to obtain $(\{(l, E_{\mathsf{Anon},l})\}_{l\in[n]}, \eta)$ and determine $E_{\mathsf{Anon},k}$ such that $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{P}||\eta)$. P_i publishes ciphertext $C_{i,k} \leftarrow m + \mathsf{sk}_{D,i} \cdot E_k$ revealing indices i, k. Notice that this ciphertext has exactly the same structure as the ciphertexts used in DHPVSS.
- **Decryption:** To decrypt a ciphertext $C_{i,k}$ for role P in slot sl, P_j checks that its $\mathsf{sk}_{L,j}$ is such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,j}) = 1$. If yes, it obtains the sender's public key $\mathsf{pk}_{D,i}$ from the blockchain and computes $m \leftarrow C_{i,k} - \mathsf{sk}_j \cdot \mathsf{pk}_{D,i}$. Notice that a proof of correct decryption can be done

exactly as in DHPVSS and that such a proof constitutes an AfP of m on behalf of role P in slot sl, since it requires proving knowledge of $\mathsf{sk}_{L,j}$ s.t. $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,j}) = 1$.

Using this slight modification of our ECW, we can instantiate DHPVSS (Figures 7 and 8) and its resharing protocol (Figure 10). The ciphertexts output by ECW have the same structure as those used in DHPVSS, so the efficient proofs of encrypted (re)share validity can be performed exactly in the same way.

Privacy and Resharing: Notice, however, that since the dealer's identity must be known when decrypting ciphertexts, using these optimized techniques for resharing will be problematic, since it requires linking a party P_i to its key $\mathsf{sk}_{D,i}$ and revealing its identity. In order to solve this issue, we can resort to a similar setup used for the regular keys E_i, *i.e.* we can allow parties access to an ideal mixnet that is used to create a shuffled set of keys $\{(j, \mathsf{sk}_{D,\mathsf{Anon},j}) : j \in [n]\}$. Now a sender can include the index to its key $\mathsf{sk}_{D,\mathsf{Anon},j}$ in the ciphertext in order to allow for decryption. As it is the case with our simple AfP technique, this would require setting up multiple such vectors, which can potentially be solved by techniques similar to those we describe in Appendix D. We leave a concrete description of such a construction for future works.

References

1. Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Ring signatures: Logarithmic-size, no setup - from standard assumptions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 281–311. Springer, Heidelberg, May 2019.
2. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, November 2020.
3. Fabrice Boudot and Jacques Traoré. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In Vijay Varadharajan and Yi Mu, editors, *ICICS 99*, volume 1726 of *LNCS*, pages 87–102. Springer, Heidelberg, November 1999.
4. Elette Boyle, Saleet Klein, Alon Rosen, and Gil Segev. Securing abe's mix-net against malicious verifiers via witness indistinguishability. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 274–291. Springer, Heidelberg, September 2018.
5. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 234–264. Springer, Heidelberg, March 2016.
6. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.
7. Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. Cryptology ePrint Archive, Report 2021/1423, 2021. `https://eprint.iacr.org/2021/1423`.
8. Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 537–556. Springer, Heidelberg, July 2017.
9. Ignacio Cascudo and Bernardo David. ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 311–341. Springer, Heidelberg, December 2020.
10. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.
11. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.

12. Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 32–46. Springer, Heidelberg, May / June 1998.
13. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.
14. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.
15. Lydia Garms, Siaw-Lynn Ng, Elizabeth A. Quaglia, and Giulia Traverso. Anonymity and rewards in peer rating systems. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 277–297. Springer, Heidelberg, September 2020.
16. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.
17. Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. Cryptology ePrint Archive, Report 2021/1397, 2021. `https://eprint.iacr.org/2021/1397`.
18. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC*, 2021.
19. Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Heidelberg, November 2017.
20. Somayeh Heidarvand and Jorge L. Villar. Public verifiability from pairings in secret sharing schemes. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 294–308. Springer, Heidelberg, August 2009.
21. Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 325–335. Springer, Heidelberg, July 2004.
22. Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999*, volume 1758 of *LNCS*, pages 184–199. Springer, Heidelberg, August 1999.
23. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.
24. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Heidelberg, December 2001.
25. A. Ruiz and J. L. Villar. Publicly verifiable secret sharing from paillier's cryptosystem. In *WEWoRC 2005– Western European Workshop on Research in Cryptology*, 2005.
26. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.
27. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 148–164. Springer, Heidelberg, August 1999.
28. Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.

A Basic notions on public key encryption

In this section we introduce well-known concepts on public key encryption.

A.1 Definitions

Definition 14. *A public key encryption scheme $\mathcal{E}$ consists of three polynomial time algorithms* $(\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ *as follows:*

- $\mathcal{E}.\mathsf{Gen}(\lambda)$ *is a probabilistic algorithm that outputs a pair* $(\mathsf{sk}, \mathsf{pk})$ *consisting of a secret key and a public key.*
- $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M)$ *is a probabilistic algorithm that takes as input a public key* pk *and a plaintext message M in a plaintext message space $\mathfrak{P}$ and outputs a ciphertexts C in a ciphertext space $\mathfrak{C}$. In addition, we will, by a slight abuse of notation, consider the function* $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M;\rho)$ *that specifies the result of* $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M)$ *when randomness ρ (in a randomness space $\mathfrak{R}$) is used.*
- $\mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(C)$ *is a deterministic function that takes secret key* sk*, and a ciphertext $C \in \mathfrak{C}$ and outputs a plaintext message $M' \in \mathfrak{P}$.*

and which satisfy that for every $(\mathsf{pk}, \mathsf{sk})$ *output by* $\mathcal{E}.\mathsf{Gen}$*, and for every $M \in \mathfrak{P}$,*

$$\Pr[\mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\mathcal{E}.\mathsf{Enc}_{pk}(M))] = 1$$

The most well known notion of security for a public key encryption scheme is IND-CPA security, which requires that the encryptions of two messages under any public key pk are computationally indistinguishable without the knowledge of the corresponding sk. Here we consider the notion of ℓ-multi-key IND-CPA security. This requires that the encryptions of two vectors of messages of the same length, where each coordinate is encrypted under a public key pk_i, are indistinguishable. The notions are equivalent as long as ℓ is polynomial in the security parameter. Since in the case of public key encryption, the adversary can always encrypt messages under the public keys pk_i and any other public key they generate, we omit that oracle in the definition.

Definition 15. *A public key encryption scheme $\mathcal{E}$ satisfies ℓ-multi-key IND-CPA security if for any PPT adversary $\mathcal{B}$, there exists a negligible function $\mu(\lambda)$ such that*

$$\left|\Pr\left[\mathrm{Game}_{\mathcal{B},\mathcal{E}}^{\ell\text{-}\mathsf{IND\text{-}CPA},0}(\lambda) = 1\right] - \Pr\left[\mathrm{Game}_{\mathcal{B},\mathcal{E}}^{\ell\text{-}\mathsf{IND\text{-}CPA},1}(\lambda) = 1\right]\right| \leq \mu(\lambda)$$

Algorithm 31 $\mathrm{Game}_{\mathcal{B},\mathcal{E}}^{\ell\text{-}\mathsf{IND\text{-}CPA},b}(\lambda)$

$\forall i \in [\ell] \quad (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathcal{E}.\mathsf{KeyGen}(pp, i)$
$(m_1^{(0)}, \ldots, m_\ell^{(0)}), (m_1^{(1)}, \ldots, m_\ell^{(1)}) \in \mathfrak{P}^\ell \leftarrow \mathcal{B}(pp, \{\mathsf{pk}_i : i \in [\ell]\})$
$\forall i \in [\ell], c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}_i}(m_i^{(b)})$
$b' \leftarrow \mathcal{B}(\{c_i : i \in [\ell]\})$
return b'

The case $\ell = 1$ is the usual IND-CPA definition and for $\ell = \mathsf{poly}(\lambda)$, a standard hybrid argument shows that a scheme is ℓ-multi-key IND-CPA if and only if it is IND-CPA.

A.2 El Gamal encryption scheme

In this paper we use El Gamal scheme, a very well known encryption scheme where the plaintext space is $\mathfrak{P} = \mathbb{G}$, a cyclic group of order p generated by G, the randomness space is $\mathfrak{R} = \mathbb{Z}_p$ and the ciphertext space is $\mathfrak{C} = \mathbb{G}^2$. The scheme $\mathcal{E}$ is given by

- $\mathcal{E}.\mathsf{Gen}(\lambda)$: Selects $\mathsf{sk} \in \mathbb{Z}_p$ uniformly at random, sets $\mathsf{pk} = \mathsf{sk} \cdot G$, outputs $(\mathsf{sk}, \mathsf{pk})$.
- $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M)$ where $M \in \mathbb{G}$, selects $\rho \in \mathbb{Z}_p$ at random, outputs $C = (\rho \cdot G, M + \rho \cdot \mathsf{pk})$ (we will denote $C = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M; \rho)$).
- $\mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(C)$, where $C = (C_1, C_2) \in \mathbb{G}^2$, outputs $\mathsf{Dec}_{\mathsf{sk}}(C) = C_2 - \mathsf{sk} \cdot C_1$.

The El Gamal encryption scheme is well known to be IND-CPA secure under the DDH assumption.

B Execution Model for Proof-of-Stake (PoS) Blockchains

In this section, we give an overview of the framework from [19] for arguing about PoS blockchain protocol security as presented in [7].

Blockchain Protocol Execution Let the blockchain protocol

$$\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$$

be guarded by a validity predicate V. The algorithms can be described as follows:

- $\mathsf{UpdateState}(1^\lambda) \to \mathsf{bst}$ where bst is the local state of the blockchain along with metadata.
- $\mathsf{GetRecords}(1^\lambda, \mathsf{bst}) \to \mathbf{B}$ outputs the longest sequence $\mathbf{B}$ of valid blocks (wrt. V).
- $\mathsf{Broadcast}(1^\lambda, m)$ Broadcast the message m over the network to all parties executing the blockchain protocol.

An execution of a blockchain protocol Γ^V proceeds by participants running the algorithm $\mathsf{UpdateState}^V$ to get the latest blockchain state, $\mathsf{GetRecords}$ to extract the ledger data structure from a state and $\mathsf{Broadcast}$ to distribute messages which are added to the blockchain if accepted by V. An execution is orchestrated by an environment $\mathcal{Z}$ which classifies parties as either honest or corrupt. All honest parties executes $\Gamma^V(1^\lambda)$ with empty local state bst and all corrupted parties are controlled by the adversary $\mathcal{A}$ who also controls network including delivery of messages between all parties.

- In each round all honest parties receive a message m from $\mathcal{Z}$ and potentially receive incoming network messages delivered by $\mathcal{A}$. The honest parties may do computation, broadcast messages and/or update their local states.
- $\mathcal{A}$ is responsible for delivering all messages sent by honest parties to all other parties. $\mathcal{A}$ cannot modify messages from honest parties but may delay and reorder messages on the network.
- At any point $\mathcal{Z}$ can communicate with adversary $\mathcal{A}$ or use $\mathsf{GetRecords}$ to retrieve a view of the local state of any party participating in the protocol.

The result is a random variable $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ denoting the joint view of all parties (i.e. all inputs, random coins and messages received) in the above execution. Note that the joint view of all parties fully determines the execution. We define the view of the adversary as $\mathsf{view}_\mathcal{A}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$

and the view of the party P_i as $\mathsf{view}_{P_i}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$. If it is clear from the context which execution the argument is referring to, then we just write view_i. We assume that it is possible to take a snapshot i.e. a view of the protocol after the first r rounds have been executed. We denote that by $\mathsf{view}^r \leftarrow \mathsf{EXEC}_r^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$. Furthermore, we can resume the execution departing from this view and continue until round $\tilde{r}$ resulting in the full view including round $\tilde{r}$ denoted by $\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}_{(\mathsf{view}^r, \tilde{r})}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$.

We let the function $\mathsf{stake}_i = \mathsf{stake}(\mathbf{B}, i)$ take as input a local blockchain $\mathbf{B}$ and a party P_i and output a number representing the stake of party P_i wrt. to blockchain $\mathbf{B}$. Let the sum of stake controlled by the adversary be $\mathsf{stake}_{\mathcal{A}}(\mathbf{B})$, the total stake held by all parties $\mathsf{stake}_{\mathsf{total}}(\mathbf{B})$ and the adversaries relative stake is $\mathsf{stake\text{-}ratio}_{\mathcal{A}}(\mathbf{B})$. We also consider the PoS-fraction $\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell)$ as the amount of unique stake whose proof is provided in the last ℓ mined blocks. More precisely, let $\mathcal{M}$ be the index i corresponding to miners P_i of the last ℓ blocks in $\mathbf{B}$ then

$$\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell) = \frac{\sum_{i \in \mathcal{M}} \mathsf{stake}(\mathbf{B}, i)}{\mathsf{stake}_{\mathsf{total}}}$$

A note on corruption For simplicity in the above execution we restrict the environment to only allow static corruption while the execution described in [23] supports adaptive corruption with erasures.

A note on admissible environments [23] specifies a set of restrictions on $\mathcal{A}$ and $\mathcal{Z}$ such that only compliant executions are considered and argues that certain security properties holds with overwhelming probability for these executions. An example of such a restriction is that $\mathcal{A}$ should deliver network messages to honest parties within Δ rounds.

Blockchain Properties In coming sections we will define what it means to encrypt to a future state of the blockchain. First, we need to ensure what it means for a blockchain execution to have evolved from one state to another. We recall that running a protocol Γ^V with appropriate restrictions on $\mathcal{A}$ and $\mathcal{Z}$ will yield certain compliant executions $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ where some security properties will hold with overwhelming probability. An array of prior works, including [14,23], have converged towards a few security properties that characterizes blockchain protocols. These include *Common Prefix* or *Chain Consistency*, *Chain Quality* and *Chain Growth*. From these basic properties, a number of stronger properties were derived in [19]. Among them, is the *Distinguishable Forking* property which will be the main requirement when introducing the EtF scheme.

Definition 16 (Common Prefix). *Let $\kappa \in \mathbb{N}$ be the common prefix parameter. The chains $\mathbf{B}_1$, $\mathbf{B}_2$ possessed by two honest parties P_1 and P_2 in slots $\mathsf{sl}_1 < \mathsf{sl}_2$ satisfy $\mathbf{B}_1^{\lceil \kappa} \preceq \mathbf{B}_2$.*

Definition 17 (Chain Growth). *Let $\tau \in (0, 1]$, $s \in \mathbb{N}$ and let $\mathbf{B}_1$, $\mathbf{B}_2$ be as above with the additional restriction that $\mathsf{sl}_1 + s \leq \mathsf{sl}_2$. Then $\mathsf{len}(\mathbf{B}_2) - \mathsf{len}(\mathbf{B}_1) \geq \tau s$ where τ is the speed coefficient.*

Definition 18 (Chain Quality). *Let $\mu \in (0, 1]$ and $\kappa \in \mathbb{N}$. Consider any set of consecutive blocks of length at least κ from an honest party's chain $\mathbf{B}_1$. The ratio of adversarial blocks in the set is $1 - \mu$ where μ is the quality coefficient.*

Definition 19 (Distinguishable Forking). *A blockchain protocol Γ satisfies $(\alpha(\cdot), \beta(\cdot), \ell_1(\cdot), \ell_2(\cdot))$-distinguishable forking property with adversary $\mathcal{A}$ in environment $\mathcal{Z}$, if there exists negligible functions, negl$(\cdot)$, $\delta(\cdot)$ such that for every $\lambda \in \mathbb{N}, \ell \geq \ell_1(\lambda), \tilde{\ell} \geq \ell_2(\lambda)$ it holds that*

$$\Pr\left[\begin{array}{c} \alpha(\lambda) + \delta(\lambda) < \beta(\lambda) \wedge \\ \mathsf{suf\text{-}stake\text{-}contr}^{\tilde{\ell}}(\mathsf{view}, \beta(\lambda)) = 1 \wedge \\ \mathsf{bd\text{-}stake\text{-}fork}^{(\ell,\tilde{\ell})}(\mathsf{view}, \alpha(\lambda) + \delta(\lambda)) = 1 \end{array} \middle| \, \mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \right]$$

$$\geq 1 - \mathrm{negl}(\lambda).$$

C Proofs for ECW

In this section we list the proofs related to theorems stated in Section 3. We re-state the theorems for convenience.

Theorem 5 (IND-CPA ECW). *Let $\mathcal{E}$ be an IND-CPA secure $\mathbb{Z}_p$-linearly homomorphic encryption scheme. The construction in Figure 4 with lottery predicate as in Section 3.1 is an IND-CPA secure ECW (as in Definition 6).*

Proof (Sketch). An adversary with a noticable advantage in $\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ described in Definition 6 can distinguish between ECW encryptions of two different messages without winning the lottery for that specific sl and P. This adversary can, in turn, distinguish between corresponding encryptions from the underlying $\mathbb{Z}_p$-linearly homomorphic encryption scheme $\mathcal{E}$, which contradicts IND-CPA security of $\mathcal{E}$. Thus, the protocol in Figure 4 yields an IND-CPA secure ECW.
IND-CCA security for the ECW scheme can be obtained by using standard transformations ([13,26]) as argued in [7].

Theorem 6 (EUF-CMA AfP). *Let $\mathcal{E}$ be an IND-CPA secure and $\mathbb{Z}_p$-linearly homomorphic encryption scheme and let* SoK *be a simulatable and extractable SoK scheme. The construction in Figure 5 with lottery predicate as in Section 3.1 is EUF-CMA AfP as defined in Definition 7.*

Proof (Sketch). We argue that an adversary who forges a signature (AfP tag) on a message m is able to construct a valid SoK on a message without knowing the witness. More precisely, assume that the adversary can make the verifier output $b = 1$ on input $(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \sigma, m)$ while not having won the lottery for parameters $(\mathbf{B}, \mathsf{sl}, \mathsf{P})$. The underlying SoK σ_{SoK} must be a convincing SoK on m such that $\mathsf{SoK.verify}((\mathbf{B}, \mathsf{sl}, \mathsf{P}), \sigma_{\mathsf{SoK}}, m) = 1$. Thus, the adversary has successfully created a SoK signature where the verification algorithm accepts but without the adversary knowing a witness. This breaks existential unforgeability of the SoK scheme contradicting our assumption.[6]

Theorem 7 (AfP Privacy). *Assume $\mathcal{E}$,* lottery *and* SoK *scheme as in 6. The construction in Figure 5 has AfP privacy as in Definition 8.*

[6]In fact, forging a signature in the EUF-CMA game of SoK reduces to either breaking the corresponding simulatability or the extractability of the SoK scheme (see [10])

Proof (Sketch). We construct a simulator $\mathcal{S}$ for the game $\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ as follows. When $\mathcal{S}$ gets a request for given tuple $(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m)$ it forwards the request to the simulator for the SoK scheme. The SoK simulator can forge a signature and, in particular, it can simulate an SoK on m without knowing the lottery winning secret key. Then, $\mathcal{S}$ obtains the response of the SoK simulator forwards it to the adversary. We claim that any adversary who can successfully distinguish between interacting with the simulator $\mathcal{S}$ and the oracle $\mathcal{O}_{\mathsf{AfP}}$ in $\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ breaks the simulatability of the SoK scheme.

D Zero knowledge proof of membership to an anonymous committee

When encryption towards a committee $\mathcal{R}$ is used as part of a protocol, identities ID_i such that $\psi(i) \in \mathcal{R}$ will typically need to act upon having received an encrypted message. This will reveal the fact that they are a receiver. However, in this section we present strategies that allow that these parties do not need to give away more than that. I.e, ID_i reveals $\psi(i) \in \mathcal{R}$ but nothing else. Note that just having parties prove knowledge of the encrypted message is not enough: a legitimate receiver ID_i may be colluding with some other party ID_{i_*} with $\psi(i_*) \notin \mathcal{R}$ and hence ID_{i*} would also be able to claim knowledge of the message. In general, we want to avoid that a set of t colluding parties of which $t' < t$ belong to $\mathcal{R}$ can claim to have $t' + 1$ or more parties in that set.

We present two solutions that each allows to solve the two problems above. The first solution is more generic but less efficient: each party in $\mathcal{R}$ signs a message using a linkable ring signature [21]. Ring signatures [24] guarantee that the signer belongs to a given set of parties without revealing their identity within that set. Linkability ensures that, despite this anonymity, two signatures using the same public key can be linked. This means colluding parties cannot use the same secret key to claim that both belong to $\mathcal{R}$, when only one of them does.

However, linkable rings signatures become larger as the size of the committee grows. In Section D.2, we present an optimized solution where we leverage the fact that, in our situation, there is already a sender broadcasting ciphertexts, and we can use this party to send auxiliary information that allows to reduce the amount of communication by each receiver to be constant-size (while the information sent by the sender is still linear in the size of the receiver committee). Our solution is based on a linkable version of Camenisch-Lysyanskaya signatures.

D.1 Generic proofs of membership based on linkable ring signatures

Ring signatures, also called sometimes Spontaneous Anonymous Group signatures, are signature schemes in which each member of a universe of parties has a secret key, which it can use to sign a message on behalf of any subset of that universe to which it belongs, in such a way that the signature does not reveal which of the parties in that subset has signed.

Ring signatures can be constructed as non-interactive zero knowledge proofs of knowledge of a secret key corresponding to a set of public keys (which is in turn an OR statement), via a Fiat-Shamir transformation where the message is included as an argument to the random oracle. In fact it is this proof of knowledge what we really need in our problem. We present the solution in terms of ring signatures because the notion of linkability is commonly used in this context. A linkable ring signature is one where if two signatures (even of different messages) for the same set of users are produced using the same secret key, this fact is detected, yet the identity of the signer is kept anonymous.

Definition 20 (Linkable Ring Signature). *A Linkable Ring Signature scheme for a set* $[n]$ *is given by the following tuple of algorithms:*

– $\mathsf{KeyGen}(n, 1^\lambda)$*: Outputs* n *key pairs* $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]}$
– $\mathsf{LinkSig}(\mathsf{sk}_i, m, \mathcal{R})$*: Takes a secret key, a message* m*, and a set* $\mathcal{R} \subseteq [n]$*, outputs a signature* σ
– $\mathsf{LinkVer}(\{\mathsf{pk}_i\}_{i\in\mathcal{R}}, m, \sigma, \mathcal{R})$*: Takes a set* $\mathcal{R} \subseteq [n]$*, a set of associated public keys* pk_i*,* $i \in \mathcal{R}$*, a message* m *and a signature* σ *and outputs* `accept` *or* `reject`
– $\mathsf{Link}((m, \sigma, \mathcal{R}), (m', \sigma', \mathcal{R}'))$*: Takes two tuples consisting of a message, a signature and a subset of* $[n]$ *and outputs a bit* b *(meant to represent whether these two signatures have been created with the same secret key)*

such that for all m, m_0, m_1*, for all sets* $\mathcal{R}, \mathcal{R}_0, \mathcal{R}_1 \subseteq [n]$*,* $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]}$ *output by* $\mathsf{KeyGen}(n, 1^\lambda)$ *and* $\mathsf{sk}, \mathsf{sk}^{(0)}, \mathsf{sk}^{(1)} \in (\mathsf{sk}_i)_{i\in[n]}$

$$\Pr[\mathsf{LinkVer}(\{\mathsf{pk}_i\}_{i\in\mathcal{R}}, m, \sigma, \mathcal{R}) = \mathit{accept})|\sigma = \mathsf{LinkSig}(\mathsf{sk}_i, m, \mathcal{R}) \wedge i \in \mathcal{R}] = 1$$

$$\Pr[\mathsf{Link}((m_0, \sigma_0, \mathcal{R}_0), (m_1, \sigma_1, \mathcal{R}_1)) = 1|\sigma_b = \mathsf{LinkSig}(\mathsf{sk}, m_b, \mathcal{R}_b), b \in \{0, 1\}] = 1$$

$$\Pr[\mathsf{Link}((m_0, \sigma_0, \mathcal{R}_0), (m_1, \sigma_1, \mathcal{R}_1)) = 0|\sigma_b = \mathsf{LinkSig}(\mathsf{sk}^{(b)}, m_b, \mathcal{R}_b), b \in \{0, 1\} \\ \wedge\ \mathsf{sk}^{(0)} \neq \mathsf{sk}^{(1)}] = 0$$

We will need several security properties from linkable ring signatures. Informally we want the following properties based on the model in [1].

– **Linkability** This requirement ensures that signatures from the same secret key will always be linked. In the game, the adversary must output k public keys for corrupted parties, and $k+1$ valid signatures, each on a message and a ring. They win if all rings are subsets of the set of the k corrupted public keys, and none of the signatures are linked.
– **Linkable Anonymity** Linkable ring signatures are publicly linkable, however a signature still should not be able to be traced to the signer's public key. In the game, the adversary is given access to an oracle to create honest users and receive their public keys. They return two honest users (their challenged users), as well as a set of their own corrupted public keys. They are then given access to an oracle, where they can submit a challenged user, a message and a ring that must contain the public keys of both challenged users. If $b = 0$, they are returned with a signature signed with the secret key of the user they input. If $b = 1$, they are returned with a signature signed by the other challenged user. They must guess b correctly to win.
– **Non–Frameability** This requirement ensures that an adversary cannot frame an honest user by forging a signature which links to this user's signature. In the game we give the adversary access to oracles to create honest users, obtain their signatures and corrupt them. The adversary must output a valid signature that was not output by the signing oracle. They then must output another valid signature that was output by the signing oracle for an honest user that has not been corrupted. For the adversary to win, the two signatures must be valid and linked.

Note that linkability implies the usual existential unforgeability security property, in the sense that, if the adversary knows no secret key $\mathsf{sk}_j, j \in \mathcal{R}$ (i.e. $|\mathcal{C} \cap \mathcal{R}| = 0$) then the adversary cannot create a valid signature for $\mathcal{R}$.

This tool almost automatically gives a solution to our problem. Each party includes a public key $\mathsf{pk}_{\mathsf{LinkSig}}$ for a linkable signature in the public key to be shuffled. To prove membership to $\mathcal{R}$,

ID_i signs a message with $\mathsf{pk}_{\mathsf{LinkSig},j}$ and publishes the message and signature. This signature can be verified by any public verifier. The security properties of the linkable signature guarantee both that the proof only reveals membership to $\mathcal{R}$ but nothing else, and that if two identities use the same secret key to claim membership to $\mathcal{R}$, this is detected by any public verifier.

One (easily fixable) caveat is that the properties above do not prevent replay attacks, where an adversary attempts to copy an honest party's signature and claim it as theirs, or at least invalidate the honest party's signature. We fix this by including the identity of the signer as a part of the message signed. We describe the construction in Figure 12.

Proof of membership verification

Set-up:
A list of anonymized public keys $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$ where $\mathsf{pk}_{\mathsf{Anon},j}$ contains a public key $\mathsf{pk}_{\mathsf{LinkSig},j}$ for a linkable signature scheme.
For the sake of notation simplicity, let $\mathsf{pk}_j := \mathsf{pk}_{\mathsf{LinkSig},j}$, $\mathsf{sk}_j := \mathsf{sk}_{\mathsf{LinkSig},j}$ in this Figure. Moreover, given $\mathcal{R} \subseteq [n]$, let $\mathcal{PK}_{\mathcal{R}} = \{\mathsf{pk}_j : j \in \mathcal{R}\}$.

Proof:
Input: A subset $\mathcal{R} \subseteq [n]$, a session identifier ssid, the key sk_j known to ID_i.
1. Set $m = H(ID_i||\mathsf{ssid})$ and output $\sigma = \mathsf{LinkSig}(\mathsf{pk}_j, m, \mathcal{R})$.

Verification:
Input: Public verifier has input $\mathcal{R}, \mathcal{PK}_{\mathcal{R}}$, ssid, and a list of pairs (ID_i, σ_i), $i \in I$ for a subset $I \subseteq \mathcal{P}$.
1. Let $m_i = H(ID_i||\mathsf{ssid})$ for $i \in I$
2. Compute the set $I' \subseteq I$ of all i in I such that $\mathsf{LinkVer}(\mathcal{PK}_{\mathcal{R}}, m_i, \sigma_i, \mathcal{R}) = \mathtt{accept}$.
3. For all $i_1, i_2 \in I'$, $i_1 \neq i_2$, if $\mathsf{Link}((m_{i_1}, \sigma_{i_1}, \mathcal{R}), (m_{i_2}, \sigma_{i_2}, \mathcal{R})) = 1$ then remove i_1, i_2 from I'. Continue until there are no such index pairs.
4. Output the remaining set I' as the set of accepted membership claims.

Fig. 12. Proof of membership to an anonymous committee

We require that if a set I of users have all generated proofs of membership to an anonymous committee honestly, then verification will pass. This is clearly true, due to the correctness of linkable ring signatures. We require three security requirements for our proof of membership to an anonymous committee:

- **Unforgeability** This requirement ensures that proofs of membership from the same party in an anonymous committee can be linked. In the game, the adversary has corrupted t parties in a anonymous committee $\mathcal{R}$ of size R. They can see proofs of membership from honest parties and must output $t+1$ proofs of memberships on the corrupted identities, i.e. for ID_i such that i has been corrupted. They win if these proofs of memberships pass verification.
  Clearly this is true for our construction in Figure 12, due to the linkability requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the unforgeability game for proofs of membership to an anonymous committee, we can win in the linkability game for linkable ring signatures. The adversary in our unforgeability game provides us with t public keys corresponding to corrupted users and we generate ourselves $R-t$ secret/ public keypairs corresponding to honest users. We can then honestly generate $R-t$ proofs of membership on behalf of honest users to provide to the adversary. They return

$t+1$ proofs of membership on behalf of corrupted users. In the linkability game we output all R public keys of all honest and corrupted members of the anonymous committee, and all $R+1$ proofs of memberships on behalf of corrupted and honest members of the anonymous committee. As all proofs of membership pass verification, we have output $R+1$ valid ring signatures that are all unlinked. Therefore, we have broken the linkability of linkable ring signatures.

- **Anonymity** Although a proof of membership of an anonymous committee reveals that the prover is a member of $\mathcal{R}$, we need to ensure that it does not reveal which member of $\mathcal{R}$. In the game, the adversary chooses two honest users in $\mathcal{R}$ and has corrupted all other users. They then receive a proof of membership on behalf of one of the honest users, and must guess which user correctly to win.

  Clearly this is true for our construction in Figure 12, due to the linkable anonymity requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the anonymity game for proofs of membership to an anonymous committee, we can win in the linkable anonymity game for linkable ring signatures. We first of all create two honest users in the linkable anonymity game. We can set the public keys of the two honest users, chosen by the adversary in the anonymity game for proofs of membership, to be these two public keys. We then submit to the challenge oracle one of these honest users, along with a ring containing all public keys in the anonymous committee and a message set to be $H(ID, \mathsf{ssid})$. We return the resulting ring signature as our proof of membership in the anonymity game, and finally return the resulting bit b output by the adversary. If the adversary wins in the anonymity game, we clearly win in the linkable anonymity game, which is a contradiction.

- **Non–Frameability** This requirement ensures that an adversary cannot frame an honest user by forging a proof of membership which links to this user's proof of membership, therefore implying unfairly that they cheated. In the game, we give the adversary access to the public keys of honest users, and oracles to obtain their proofs of membership. The adversary must output a proof of membership that was not output by the oracle. They then must output another proof of membership that was output by the signing oracle for an honest user. For the adversary to win, the two signatures must not pass verification together, but should pass verification individually.

  Clearly this is true for our construction in Figure 12, due to the non–frameability requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the non–frameability game for proofs of membership to an anonymous committee, we can win in the non–frameability game for linkable ring signatures. We will provide the adversary in the non–frameability game for proofs of membership with the public keys of honest users, using the corresponding oracle in the non–frameability game for linkable ring signatures. When the adversary in the non–frameability game for proofs of membership attempts to obtain the proofs of memberships for honest users, we will use the signing oracle in the non–frameability game for linkable ring signatures. The adversary in the non–frameability game for proofs of membership will output two proofs of membership that individually pass verification, but fail together: one output from the signing oracle for an honest user and one that was not output by the signing oracle. We can then output these two proofs of membership in the linkable ring signature game. They will both be valid and linked signatures, so we will win in the non-frameability game for linkable ring signatures.

D.2 Efficient instantiation using Camenisch-Lysyanskaya signatures

In this section, we propose a solution where the size of a membership proof is constant (independent from the size of $\mathcal{R}$). For this we leverage the fact that the sender can send auxiliary information together with the ciphertexts. Our strategy is based on a "linkable version" of a signature scheme by Camenisch-Lysyanskaya.

We focus on one version of the Camenisch-Lysyanskaya signatures which has been used for anonymous credentials and where we want to construct a signature of a group element $sG \in \mathbb{G}$. A crucial feature of this proof is that it can be divided in two parts: the first part uses the signing key and does not require knowledge of s and outputs σ; meanwhile, the second part is a proof of knowledge of s and does not require to know the secret signing key.

This means that the two parts of the proof can be carried out by two different parties. Moreover the signature has a second important property: if the owner of the signature key has carried out the first part of the signing for different s_iG, with outputs σ_i, then the second part of the signature (the proof of knowledge) does not reveal which σ_i is being completed.

Our strategy is then the following. The sender carries out the first part of the CL signature of each of the public keys of the parties in $\mathcal{R}$, thereby creating messages σ_i. Now, because of what we mentioned above, any receiver can prove the knowledge of the discrete logarithm of one of these secret keys, without revealing which.

As before, this has the problem that, if a party ID_i in $\mathcal{R}$ is colluding with other parties outside the set, then they could all use the secret key known by ID_i and claim to be in $\mathcal{R}$. In order to prevent that we turn the signature into a linkable one by including another generator H in the common reference string, and having each receiver publish $I_j = \mathsf{sk}_{\mathsf{Anon},j}H$. We extend the proof of knowledge of $\mathsf{sk}_{\mathsf{Anon},j}$ into one that ensures $\mathsf{sk}_{\mathsf{Anon},j}$ is the same as the discrete log of I_j in base H. Since I_j is deterministically computed from H and $\mathsf{sk}_{\mathsf{Anon},j}$, a verifier can easily check if two parties have claimed the same key.

Camenisch-Lysyanskaya signatures The precise signature we will use is the one called Signature A in [6], but with the difference that while that paper assumed a type I bilinear pairing (which would not allow for using the DDH assumption), we will replace it by a Type III bilinear pairing as has been done in other works such as [5,15].

We recall this signature scheme: Let $\mathbb{G}_1, \mathbb{G}_2$ (with additive notation) and $\mathbb{G}_T$ (with multiplicative notation) be groups of prime order p. Let $\mathbb{G}_1$ be generated by G_1 and $\mathbb{G}_2$ be generated by G_2. The signing secret key is of the form $\mathsf{sk}_{\mathsf{CL}} = (x, y) \in \mathbb{Z}_p^2$ and the public key $\mathsf{pk}_{\mathsf{CL}} = (X, Y) = (xG_2, yG_2)$ in $\mathbb{G}_2^2$.

The signature scheme can be used to either sign messages $m \in \mathbb{Z}_p$ or $M = mG_1 \in \mathbb{G}_1$. We are interested in the latter case. As mentioned above, this case can be separated in two algorithms, where $\mathsf{CL.Sig}^1$ uses M and $\mathsf{sk}_{\mathsf{CL}}$ but does not require knowledge of m, and $\mathsf{CL.Sig}^2$ is applied to the output of $\mathsf{CL.Sig}^1$ and requires knowledge of m, but not of the secret key. These protocols are defined in Figure 13.

A crucial point is that the verification step depends only on the output of $\mathsf{CL.Sig}^2$. Moreover, given $(M_1, \sigma_1^1), \ldots, (M_n, \sigma_n^1)$ where $M_i = m_iG_1$ and $\sigma_i^1 = \mathsf{CL.Sig}^1{}_{(x,y)}(M_i)$, the signature $\mathsf{CL.Sig}^2(m_i, \sigma_i^1)$ gives no information about i

This means that, once $\mathsf{CL.Sig}^1$ has been carried out on the messages M_i, the second step of the signature can be seen as a ring signature scheme of sorts: if we interpret (m_i, M_i) as a secret key/public key pair belonging to the i-th party in a given set of parties, as it will be our case, then

Camenisch-Lysyanskaya signature

Setup: Groups $(\mathbb{G}_1, +), (\mathbb{G}_2, +), (\mathbb{G}_T, \cdot)$ of order p with generators G_1, G_2 for $\mathbb{G}_1$, $\mathbb{G}_2$ respectively, bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A random oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$.

Parties and keys: A sender has a CL keypair ($\mathsf{sk}_{\mathsf{CL}} = (x, y) \in \mathbb{Z}_p^2, \mathsf{pk}_{\mathsf{CL}} = (X, Y)$) where $X = xG_2$, $Y = yG_2$.

Algorithm 32 $\mathsf{CL.Sig}^1{}_{\mathsf{sk_{CL}}}(M)$

parse $(x, y) \leftarrow \mathsf{sk}_{\mathsf{CL}}$
$a \leftarrow\$\, \mathbb{Z}_p$, $A \leftarrow aG_1 \in \mathbb{G}_1$
$B \leftarrow yA$, $C \leftarrow xA + axyM$
return $\sigma^1 \leftarrow (A, B, C)$ ▷ Note that if we call $M = mG$, then $C = xA + axyM = (x + mxy)A$.

Algorithm 33 $\mathsf{CL.Sig}^2(m, \sigma^1)$

Parse σ^1 as (A, B, C)
$r, r' \leftarrow\$\, \mathbb{Z}_p^*$
$\tilde{A} \leftarrow r'A$, $\tilde{B} \leftarrow r'B$, $\hat{C} \leftarrow rr'C$
$z_A \leftarrow e(\tilde{A}, X)$, $z_B \leftarrow e(\tilde{B}, X)$, $z_C \leftarrow e(\hat{C}, G_2)$
$\rho \leftarrow r^{-1}$
$\mathcal{W} \rightarrow \mathbb{Z}_p^2$, $\mathcal{X} \leftarrow \mathbb{G}_T$, $pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
$\pi \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}((\rho, m); pp_\pi, z_A, f_{(z_B, z_C)})$
where $f_{(z_B,z_C)}(\rho, m) = z_B^{-m} z_C^{\rho}$. ▷ This proves knowledge of ρ and m such that $z_B^{-m} z_C^{\rho} = z_A$.
return $\sigma^2 \leftarrow (\tilde{A}, \tilde{B}, \hat{C}, \pi)$

Algorithm 34 $\mathsf{Ver}^2(\mathsf{PK}, \sigma^2)$

Parse $\sigma^2 \leftarrow (\tilde{A}, \tilde{B}, \hat{C}, \pi)$
Compute z_A, z_B, z_C as in $\mathsf{CL.Sig}^2$ above.
return accept iff $e(\tilde{A}, Y) = e(\tilde{B}, G_2)$ and $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, z_A, f_{(z_B,z_C)})$ accepts.

Fig. 13. Camenisch-Lysyanskaya signature

by executing $\mathsf{CL.Sig}^2$ on the output of $\mathsf{CL.Sig}^1{}_{(x,y)}(M_i)$ the i-th party is creating a signature (for an "empty" message) that guarantees this party belongs to the set, without revealing their identitity.

Adding linkability To ensure linkability in the scenario we just described, namely that any verifier can detect when $\mathsf{CL.Sig}^2$ has been applied twice on the same input, we do the following:

First, as part of the setup we fix H, a generator of group $\mathbb{G}_1$, as part of the set up. Then $\mathsf{CL.LinkSig}^2(m, \sigma^1)$ works as follows:

Algorithm 35 $\mathsf{CL.LinkSig}^2(m, \sigma^1)$

$I \leftarrow mH$
Compute $(\tilde{A}, \tilde{B}, \hat{C}, \pi')$ as in $\mathsf{CL.Sig}^2(m, \sigma^1)$ except now
$\pi' \leftarrow \Pi_{\mathsf{NI-Pre}}((\rho, m); pp_\pi, (z_A, I), f'_{(z_B, z_C, H)})$,
where $f'_{(z_B, z_C, H)}(\rho, m) := (z_B^{-m} z_C^{\rho}, mH)$.
and $pp_\pi = (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}', \mathcal{H})$ where $\mathcal{X}' = \mathbb{G}_T \times \mathbb{G}_1$
return $\sigma^2 \leftarrow (\tilde{A}, \tilde{B}, \hat{C}, I, \pi')$

Now I depends deterministically on m and public information, and therefore a verifier can detect if the same m is used twice, as it will yield the same I.

Final instantiation Our final instantiation, described formally in Figure 14 is as follows: The sender will encrypt the message with the El Gamal encryption scheme under the anonymous public keys in $\mathcal{R}$ and include a proof of correctness of encryption. Moreover, the sender will compute $\sigma_j^1 = \mathsf{CL.Sig}^1_{\mathsf{sk_{CL}}}(\mathsf{pk_{Anon,\mathit{j}}})$ for $j \in \mathcal{R}$, where $\mathsf{sk_{CL}}$ is the secret key for the sender. Finally, we observe that in the description of CL signatures above there is no guarantee that σ_j^1 has been computed correctly until Ver^2 is executed, so we need the sender to additionally prove that σ_j^1 is indeed computed correctly from $\mathsf{CL.Sig}^1(\mathsf{pk_{Anon,\mathit{j}}})$. To claim membership to $\mathcal{R}$, and therefore ownership of some $\mathsf{sk_{Anon,\mathit{j}}}$, a party can then compute $\sigma^2 = \mathsf{CL.Sig}^2(\mathsf{sk_{Anon,\mathit{j}}}, \sigma_j^1)$. As in the generic construction, to avoid replay attacks we add the public identity of the prover in the argument of the Fiat-Shamir random oracle for the proof of knowledge π.

The correctness of the EncAMC scheme is satisifed, due to the correctness of the Camenisch-Lysyanskaya signatures. Clearly the prooofs π_{EC} and π_{CLSC} guarantee that the sender has behaved honestly. We again require three security requirements for our proof of membership to an anonymous committee as defined previously:

- **Unforgeability**
  Clearly this is true for our construction in Figure 14, due to the LRSW assumption [22], which ensures the security of Camenisch-Lysyanskaya signatures. We now provide a proof sketch. For an adversary to have output $k+1$ proofs of memberships that pass verification and that were not honestly generated, after having corrupted t of the public keys, they must have returned $t+1$ signatures that are valid according to Ver^2, containing elements $I_1, \cdots I_{t+1}$ with for all $(i,j) \in [t+1]$ $I_i \neq I_j$. Say $\exists i \in [t+1]$ such that $I_i = \mathsf{sk}H$, where sk is the secret key of an honest user. Then we can build an adversary that can break the discrete logarithm, by extracting sk due to the proof of knowledge property. Say $\exists i \in [k+1]$ such that $I_i = \mathsf{sk}H$, where sk is not the secret key of any user (corrupt or honest). Then we can build an adversary that can break the unforgeability of CL signatures, because the adversary has forged a signature on a new message $\mathsf{sk}G$. Now it is not possible for all $I_1, \cdots, I_{t+1}$ to be distinct, as there are only t corrupted users, and so we have a contradiction.
- **Anonymity**
  Clearly this is true for our construction in Figure 14, due to the DDH assumption. We now provide a proof sketch. We show that given an adversary that can win in the anonymity game for proofs of membership to an anonymous committee, we can distinguish DDH tuples. We are input $X_1, X_2, X_3, X_4 \in \mathbb{G}_1^4$. In setup we set $G = X_1$, $H = X_2$. We choose bit $b \leftarrow\!\!\mathrm{s}\, \{0,1\}$ and set the public key of the bth honest user to be X_3. We then generate a proof of membership as follows. We set $I = X_4$, and choose $\tilde{A}$, $\tilde{B}$, $\tilde{C}$ as normal based on the signature $\sigma^1 = (A, B, C)$

Encryption to a committee with anonymous membership claim

Setup: Groups $(\mathbb{G}_1, +), (\mathbb{G}_2, +), (\mathbb{G}_T, \cdot)$ of order p. Generators G_1, H for $\mathbb{G}_1$, generator G_2 for $\mathbb{G}_2$, bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

Parties and keys: A sender has a CL keypair $(\mathsf{sk}_{\mathsf{CL}} = (x, y) \in \mathbb{Z}_p^2, \mathsf{pk}_{\mathsf{CL}} = (X, Y))$ where $X = xG_2$, $Y = yG_2$. In addition, there is a set $\mathcal{P}$ of potential receivers. In the setup phase, every party chooses a keypair $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} \in \mathbb{Z}_p$, $\mathsf{pk} = \mathsf{sk}G_1 \in \mathbb{G}_1$ and then inputs it to a mix-net, resulting in a public list $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$.

Algorithm 36 $\mathsf{EncAMC.Enc}(M, \mathsf{sk}_{\mathsf{CL}}; \mathcal{R}, (\mathsf{pk}_{\mathsf{Anon},j})_{j\in\mathcal{R}})$ where $M \in \mathbb{G}_1$, $\mathcal{R} \subseteq [n]$

$\forall j \in \mathcal{R}$, $r_j \leftarrow\$\, \mathbb{Z}_p$
$\forall j \in \mathcal{R}$, $c_j \leftarrow \mathcal{E}.\mathsf{Enc}_{pk_{\mathsf{Anon},j}}(M; r_j)$
$\pi_{\mathsf{EC}} \leftarrow \mathcal{E}.\mathsf{ProveEnc}(M, (r_j)_{j\in\mathcal{R}}; (c_j)_{j\in\mathcal{R}})$,
$\forall j \in \mathcal{R}$, $\sigma_j^1 \leftarrow \mathsf{CL.Sig}^1_{\mathsf{sk}_{\mathsf{CL}}}(\mathsf{pk}_{\mathsf{Anon},j})$ with randomness $a_j \in \mathbb{Z}_p$
$\pi_{\mathsf{CLSC}} \leftarrow \mathsf{CLSC.Prove}(x, y, (a_j)_{j\in\mathcal{R}}; X, Y, (\mathsf{pk}_{\mathsf{Anon},j})_{j\in\mathcal{R}}, (\sigma_j^1)_{j\in\mathcal{R}})$,
  as in Figure 15 below (this proves that σ_j^1 are correct CLSC signatures)
return $((c_j, \sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{EC}}, \pi_{\mathsf{CLSC}})$

Algorithm 37 $\mathsf{EncAMC.Ver}((c_j, \sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{EC}}, \pi_{\mathsf{CLSC}})$

return accept iff
  both $\mathcal{E}.\mathsf{VerifyEnc}((c_j)_{j\in\mathcal{R}}, \pi_{\mathsf{EC}})$ and $\mathsf{CLSC.Verify}((\sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{CLSC}})$ accept.

Algorithm 38 $\mathsf{EncAMC.Claim}(\mathsf{sk}_{\mathsf{Anon},j}; (\sigma_j^1)_{j\in\mathcal{R}})$

return $\sigma^2 \leftarrow \mathsf{CL.LinkSig}^2(\mathsf{sk}_{\mathsf{Anon},j}, \sigma_j^1)$

Algorithm 39 $\mathsf{EncAMC.ClaimVer}(\{(\sigma_i^2)_{i\in I}\}, \mathsf{pk}_{\mathsf{CL}})$

Receive as input a set $(\sigma_i^2)_{i\in I}$ of verification claims
For all $i \in \mathcal{I}$, parse $\sigma_i^2 = (\tilde{A}_i, \tilde{B}_i, \hat{C}_i, I_i, \pi_i')$.
for each $I \in \mathbb{G}_T$ such that there are more than one $i \in \mathcal{I}$ with $I_i = I$ **do**
  Let $\mathcal{I}_I$ the set of such i.
  if there is exactly one i in $\mathcal{I}_I$ such that $\mathsf{Ver}^2(\mathsf{pk}_{\mathsf{CL}}, \sigma_i^2)$ accepts **then**
    Accept this claim and reject all other claims from parties in $\mathcal{I}_I$
  else
    Reject all membership claims from parties in $\mathcal{I}_I$
  end if
end for
for each I such that there is one $i \in \mathcal{I}$ with $I_i = I$ **do**
  Accept the claim if and only if $\mathsf{Ver}^2(\mathsf{pk}_{\mathsf{CL}}, \sigma_i^2)$ accepts
end for

Fig. 14. Encryption to an anonymous committee via CL signatures

of the bth honest user. We then simulate the attached proof, which is possible to the zero knowledge property. If the adversary guesses correctly, we output 1, and otherwise we output 0. If a DDH tuple is input, then the inputs to the adversary in the proof of membership game are distributed correctly and we output 1 with the same probability that the adversary is successful.

Proof of Camenisch-Lysyanskaya Signature Correctness CLSC

Proof for relation

$$\begin{aligned} R_{\mathsf{CLSC}} = \{((x, y, (a_j)_{j\in\mathcal{R}}); (X, Y, (\mathsf{pk}_j)_{j\in\mathcal{R}}, (\sigma^1_j)_{j\in\mathcal{R}})) :\\ \sigma^1_j = (a_j \cdot G_1, a_j \cdot y \cdot G_1, a_j \cdot x \cdot G_1 + a_j \cdot x \cdot y \cdot \mathsf{pk}_{\mathsf{Anon},j}),\\ X = x \cdot G_1,\\ Y = y \cdot G_1\} \end{aligned}$$

Algorithm 40 $\mathsf{CLSC.Prove}(x, y, (a_j)_{j\in\mathcal{R}}; X, Y, (\mathsf{pk}_j)_{j\in\mathcal{R}}, (\sigma^1_j)_{j\in\mathcal{R}})$

Let $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ random oracle.
  for j in $\mathcal{R}$ **do**
    parse $\sigma^1_j \leftarrow (\sigma^1_{j1}, \sigma^1_{j2}, \sigma^1_{j3})$.
    $b_j \leftarrow a_j y$, $c_j \leftarrow a_j x$, $d_j \leftarrow a_j xy$. ▷ Introducing these new variables "linearizes" the problem
  end for
  $\mathcal{W} \leftarrow \mathbb{Z}_p^{4|\mathcal{R}|}$, $\mathcal{X} \leftarrow \mathbb{G}_1^{6|\mathcal{R}|}$, $pp \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
  return $\pi_{\mathsf{CLSC}} \leftarrow \mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$ where
    $w = (a_j, b_j, c_j, d_j)_{j\in\mathcal{R}}$
    $x = (\sigma^1_{j1}, \sigma^1_{j2}, \sigma^1_{j3}, O, O, O)_{j\in\mathcal{R}}$
    $f(w) := (a_j G_1, b_j G_1, c_j G_1 + d_j \mathsf{pk}_j, a_j Y_D - b_j G_1, a_j X - c_j G_1, c_j Y - d_j G_1)_{j\in\mathcal{R}}$
  (Note that, for each $j \in \mathcal{R}$, the first 3 conditions in f check the target statement using the introduced variables, while the three last ensure these variables are correctly defined)

Algorithm 41 $\mathsf{CLSC.Verify}(X, Y, (\mathsf{pk}_j)_{j\in\mathcal{R}}, (\sigma^1_j)_{j\in\mathcal{R}}, \pi_{\mathsf{CLSC}})$

  return $\mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Verify}(x, f, \pi_{\mathsf{CLSC}})$ where x, f are defined from the σ^1_j as above.

Fig. 15. Proof of Camenisch-Lysyanskaya signature correctness

If a DDH tuple is not input, then the inputs to the adversary are independent of b, and we output 1 with probability 1/2.

– **Non–Frameability**

Clearly this is true for our construction in Figure 14, due to the non–frameability requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the non–frameability game for proofs of membership to an anonymous committee, we can break the discrete logarithm assumption. We are input $X_1, X_2 \in \mathbb{G}_1^2$. In setup we set $G = X_1$, $H = G^a$, where $a \leftarrow_\$ \mathbb{Z}_p$. When the adversary in the non–frameability game for proofs of membership attempts to create an honest user, we will behave normally, except for one honest user i^* where we will set $\mathsf{pk} = X_2$. When the adversary queries the oracle for proofs of memberships for this user i^*, we will set $I = \mathsf{pk}^a$, which is distributed correctly, generate $\tilde{A}, \tilde{B}$ and $\tilde{C}$ as normal for $\sigma^1 = (A, B, C)$ and simulate the proof, which is possible due to the zero knowledge property. The adversary in the non–frameability game for proofs of membership will output two proofs of membership that individually pass verification, but fail together: one generated honestly by the oracle and one that was not output by the oracle. Assume that the proof of membership output from the oracle, was that of the honest user i^*, which occurs with probability $1/k$, where k is the number of honest users. Then for both signatures output, $I = X_2^a$. We can then extract the discrete logarithm of I base H from the attached proof, due to the proof of knowledge property, which will provide the discrete logarithm of X_2 base X_1.

E Other security proofs

E.1 Security of Π_{Pre}

The Σ-protocol Π_{Pre} is obviously complete. It has special soundness because given two accepting transcripts (a, e, z), (a, e', z') with $e \neq e'$, one can extract w as $(e - e')^{-1}(z - z')$. It is therefore a proof of knowledge of w with soundness error $1/|\mathbb{F}|$. Finally it is honest-verifier zero-knowledge: a simulator can produce a transcript that is indistinguishable from a real one by choosing z uniformly at random in $\mathcal{X}$, and e uniformly at random in $\mathbb{F}$, and then computing $a = f(z) - e \cdot x$, which is uniformly random in $\mathcal{W}$.

E.2 Correctness and security of HEPVSS

Lemma 1 (Correctness of HEPVSS). *If $\mathcal{E}$ is correct then construction* HEPVSS *for a publicly verifiable secret sharing scheme satisfies* correctness*.*

Proof. Consider a set of encrypted shares $\{C_i : i \in [n]\}$ and a proof $\mathsf{Pf_{Sh}}$ output by $\mathsf{HEPVSS.Dist}$ with respect to parameters $pp = (\mathbb{G}, G, p, \{\alpha_i : i \in [n]\}, \mathcal{E})$, a secret $S = \gamma_0 \cdot G \in \mathbb{G}$, and a set of public keys $\{\mathsf{pk}_i : i \in [n]\}$ generated by $\mathsf{HEPVSS.KeyGen}$.

For all $i \in [n]$, $(A_i, \mathsf{Pf}_{\mathsf{Dec}i})$ are output by $\mathsf{HEPVSS.DecShare}(pp, \mathsf{pk}_i, \mathsf{sk}_i, C_i)$ where $\{\mathsf{sk}_i : i \in [n]\}$ are the secret keys generated by $\mathsf{HEPVSS.KeyGen}$. Then, by definition of correctness, HEPVSS is correct if

$$\mathsf{HEPVSS.Verify}(pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf_{Sh}}) = 1,$$

for all $i \in [n]$

$$\mathsf{HEPVSS.VerifyDec}(pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}i}) = 1$$

and finally $\mathsf{HEPVSS.Rec}$ outputs the secret S.

Assume that $\mathsf{HEPVSS.Verify}$ does not return 1. Then, $\mathsf{Pf_{Sh}}$ is not a valid proof. By assumption, the proofs system is correct. As for all $i \in [n]$, $C_i = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(S + m(\alpha_i) \cdot G, \rho_i)$, and m is chosen to have degree $\leq t$ with $m(\alpha_0) = 0$, then the proof $\mathsf{Pf_{Sh}}$ will be valid. Therefore, algorithm $\mathsf{HEPVSS.Verify}$ returns 1.

Assume that $\mathsf{HEPVSS.VerifyDec}(pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}i})$ does not return 1 for some $i \in [n]$. Then, $\mathsf{Pf}_{\mathsf{Dec}i}$ is not a valid proof. By assumption, the proof system is correct. Because $A_i = \mathsf{Dec}_{\mathsf{sk}_i}(C_i)$, then the proof $\mathsf{Pf}_{\mathsf{Dec}i}$ will be valid. Therefore, $\forall i \in [n]$ algorithm $\mathsf{HEPVSS.VerifyDec}$ returns 1.

Assume that $\mathsf{HEPVSS.Rec}$ does not output the secret S. $\mathsf{HEPVSS.Rec}$ will output

$$S' = \mathsf{GShamir.Rec}(pp, \{\mathsf{Dec}_{\mathsf{sk}_i}(\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(S + m(\alpha_i) \cdot G)) : i \in \mathcal{T}\}).$$

Due to the correctness of $\mathcal{E}$, $S' = \mathsf{GShamir.Rec}(pp, \{(S + m(\alpha_i) \cdot G) : i \in \mathcal{T}\})$, which will clearly output S.

Then, by contradiction, HEPVSS is correct.

Theorem 8 (IND-1 Secrecy). *If $\mathcal{E}$ is IND-CPA then construction* HEPVSS *for a PVSS satisfies* indistinguishability of secrets

Proof. Let $\mathcal{A}$ be an adversary that can win the IND-1 Secrecy Game for HEPVSS with non-negligible advantage ϵ. Note that as a dealer secret key is not needed to perform $\mathsf{HEPVSS.Dist}$, we do not need to consider the DIST oracle. We construct an adversary $\mathcal{B}$ that uses to $\mathcal{A}$ to break $(n-t)$-multi-key IND-CPA security, which is equivalent to IND-CPA security (see Appendix A).

Firstly, $\mathcal{B}$ passes the keys pk_i, $i \in [n-t]$ to $\mathcal{A}$ and observes its constructed pk_i, $i \in [n-t+1, n]$. Then $\mathcal{B}$ chooses random polynomials $m^{(0)}$, and $m^{(1)}$ of degree at most t under the restriction that $m^{(0)}(\alpha_i) = m^{(1)}(\alpha_i)$ for i in $[n-t+1, n]$. [7]

$\mathcal{B}$ sets $S_0 = m^{(0)}(\alpha_0) \cdot G$, $S_1 = m^{(1)}(\alpha_0) \cdot G$, $A_i^{(0)} = m(\alpha_i) \cdot G$, $A_i^{(1)} = m(\alpha_i') \cdot G$ for $i \in [n]$ and sends message vectors $\mathbf{m}^{(j)} = (A_1^{(j)}, A_2^{(j)}, \ldots, A_{n-t}^{(j)})$ for $j \in \{0,1\}$ to the IND-CPA challenger, and receives $(C_1, ..., C_{n-t})$ in return where $C_i = \mathsf{Enc}_{\mathsf{pk}_i}(A_i^{(b)})$.

Now $\mathcal{B}$ computes $C_i = \mathsf{Enc}(m^{(0)}(\alpha_i) \cdot G)$ for $i \in [n-t+1, n]$ and note that for these values of i, $m^{(0)}(\alpha_i) = m^{(1)}(\alpha_i)$ so $C_i = \mathsf{Enc}(m^{(1)}(\alpha_i) \cdot G)$ too. Finally given C_i, $i \in [n]$, $\mathcal{B}$ constructs a simulated proof $\mathsf{Pf_{Sh}}^*$. Now $\mathcal{B}$ sends $C_i, i = 1, \ldots, n$, and $\mathsf{Pf_{Sh}}^*$ to $\mathcal{A}$ as well as the candidate secret S_0. $\mathcal{B}$ then outputs the same guess as $\mathcal{A}$.

It is clear that $\mathcal{A}$ receives from $\mathcal{B}$ encrypted shares of S_0 (if the challenger's bit is $b = 0$) or S_1 (if $b = 1$) distributed identically as in the protocol: indeed the ciphertexts $C_1, \ldots, C_{n-t}$ are the encryptions of either the set $\{A_i^{(0)}\}_{i \in [n-t]}$ or $\{A_i^{(1)}\}_{i \in [n-t]}$ of the first $n-t$ shares constructed by $\mathcal{B}$ for the secrets, and the last t ciphertexts created by $\mathcal{B}$ are encryptions of $A_i^{(0)} = A_i^{(1)}$, $i \in [n-t+1, n]$. Finally $\mathsf{Pf_{Sh}}^*$ is computationally indistinguishable from a real proof of correct sharing by the zero knowledge property of the proof. Therefore the guessing advantage of $\mathcal{B}$ for the multi-key IND-CPA game is the same as that of $\mathcal{A}$.

Lemma 2 (Verifiability of HEPVSS). *Construction* HEPVSS *for a publicly verifiable secret sharing scheme satisfies* verifiability*.*

Proof. Verifiability of Key Generation. Our construction clearly satisfies verifiability of key generation because public keys simply consist of one group element, and so it is easy to verify public keys are correctly formed.

Verifiability of Distribution. Our construction satisfies verifiability of distribution because if $\mathsf{HEPVSS.Verify}(pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf_{Sh}}) = 1$ then $\mathsf{Pf_{Sh}}$ is a valid proof of witness $w = (S, m(X), \rho_1, \ldots, \rho_n)$ such that $m(\alpha_0) = 0$, for all $i \in [n]$ $C_i = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(S + m(\alpha_i) \cdot G, \rho_i)$ and m has degree $\leq t$. Therefore, clearly $\mathsf{HEPVSS.Dist}$ on input $pp, \{\mathsf{pk}_i : i \in [n]\}, S$ and with randomness $m(X), \rho_1, \ldots, \rho_n$ will output $\{C_i : i \in [n]\}$.

Verifiability of Decryption. Our construction clearly satisfies verifiability of decryption because if $\mathsf{HEPVSS.VerifyDec}(pp, \mathsf{pk}, A, C, \mathsf{Pf_{Dec}}) = 1$ then $\mathsf{Pf_{Dec}}$ is a valid proof of witness sk such that $A = \mathsf{Dec_{sk}}(C)$ and sk is the secret key corresponding to pk. Therefore, $\mathsf{DecShare}(pp, \mathsf{pk}, \mathsf{sk}, C) = (A, \cdot)$ for any randomness input to this algorithm.

E.3 Correctness and security of DHPVSS

Lemma 3 (Correctness of DHPVSS). *Our construction* DHPVSS *for a publicly verifiable secret sharing scheme satisfies* correctness*.*

[7] Which can be done by choosing first $m^{(0)}(X)$ and then taking $m^{(1)}(X) = m^{(0)}(X) + \gamma \cdot \prod_{i=n-t+1}^{n}(X - \alpha_i)$ for uniformly random γ.

Proof. Consider a set of encrypted shares $\{C_i : i \in [n]\}$ and a proof $\mathsf{Pf}_{\mathsf{Sh}}$ output by $\mathsf{DHPVSS.Dist}$ with respect to parameters $pp = (\mathbb{G}, G, p, \{\alpha_i, v_i : i \in [n]\})$ output by $\mathsf{DHPVSS.Setup}$, a secret $S = \gamma_0 \cdot G \in \mathbb{G}$, a public and secret key $(\mathsf{pk}_D, \mathsf{sk}_D)$ generated by $\mathsf{DHPVSS.DKeyGen}$, and a set of public keys $\{\mathsf{pk}_i : i \in [n]\} = \{(E_i = \mathsf{sk}_i \cdot G, \Omega_i) : i \in [n]\}$ generated by $\mathsf{DHPVSS.KeyGen}$ with respect to $\{id_i : i \in [n]\}$.

Clearly for all $i \in [n]$, $\mathsf{VerifyKey}(pp, id_i, \mathsf{pk}_i) = 1$, as because of the correctness of the proofs of discrete logarithms, Ω_i will be valid.

For all $i \in [n]$, $(A_i = C_i - \mathsf{sk}_i \cdot \mathsf{pk}_D, \mathsf{Pf}_{\mathsf{Dec}i})$ are output by $\mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}_i, \mathsf{sk}_i, C_i)$. Then, by definition of correctness, DHPVSS is correct if

$$\mathsf{DHPVSS.Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1,$$

for all $i \in [n]$

$$\mathsf{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i}) = 1$$

and finally $\mathsf{DHPVSS.Rec}$ outputs the secret S.

Assume that $\mathsf{DHPVSS.Verify}$ does not return 1. Then, $\mathsf{Pf}_{\mathsf{Sh}}$ is not a valid proof for G, pk_D, U, V such that

$$V = \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot C_i,$$

$$U = \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot E_i,$$

where $f^* = H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$ and $\forall i \in [n]$

$$v_i = \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}.$$

By assumption, the proofs of discrete logarithm equality are correct. As $C_i = \mathsf{sk}_D \cdot E_i + A_i$ where $A_i = S + m(\alpha_i) \cdot G$ for polynomial m of degree $\leq t$ such that $m(\alpha_0) = 0$, then

$$V = \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot C_i = \mathsf{sk}_D \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot E_i + \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot (S + m(\alpha_i) \cdot G)$$

$$= \mathsf{sk}_D \cdot U + \sum_{i=1}^{n} v_i f^*(\alpha_i)(S + m(\alpha_i) \cdot G).$$

As m is a polynomial of degree t such that $m(\alpha_0) = 0$ and due to Theorem 1, $V = \mathsf{sk}_D \cdot U$. As $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$, then the proof $\mathsf{Pf}_{\mathsf{Sh}}$ will be valid. Therefore, algorithm $\mathsf{DHPVSS.Verify}$ returns 1.

Assume that there exists $i \in [n]$ such that $\mathsf{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i})$ does not return 1. Then, $\mathsf{Pf}_{\mathsf{Dec}i}$ is not a valid proof for $G, E_i, \mathsf{pk}_D, C_i - A_i$. By assumption, the proofs of discrete logarithm equality are correct. Because $C_i - A_i = \mathsf{sk}_i \cdot \mathsf{pk}_D$ and $E_i = \mathsf{sk}_i \cdot G$, then the proof $\mathsf{Pf}_{\mathsf{Dec}i}$ will be valid. Therefore, $\forall i \in [n]$ algorithm $\mathsf{DHPVSS.VerifyDec}$ returns 1.

Assume that $\mathsf{DHPVSS.Rec}$ does not output the secret S. $\mathsf{DHPVSS.Rec}$ will output

$$S' = \mathsf{GShamir.Rec}(pp, \{A_i : i \in [n]\}).$$

The secret $\gamma_0 \cdot G = S$ will be output.

Then, by contradiction, DHPVSS is correct.

Lemma 4 (IND-1 Secrecy of DHPVSS). *Our construction* DHPVSS *for a publicly verifiable secret sharing scheme satisfies* indistinguishability of secrets *if the DDH assumption holds.*

Proof. Suppose there is an adversary $\mathcal{A}$ such that

$$\Pr\left[\text{Game}_{\mathcal{A},\mathsf{PVSS}}^{\mathsf{ind\text{-}secrecy},0}(\lambda)=1\right]-\Pr\left[\text{Game}_{\mathcal{A},\mathsf{PVSS}}^{\mathsf{ind\text{-}secrecy},1}(\lambda)=1\right]=\epsilon,$$

where ϵ is non-negligible, then we can construct $\mathcal{B}$ that distinguishes DDH tuples with non-negligible probability. We give the detailed description of $\mathcal{B}$ in Algorithm 42, and then explain how $\mathcal{B}$ works.

We now explain why, when a DDH tuple is input to $\mathcal{B}$, the view of $\mathcal{A}$, when $j=1$ and $\tilde{b}=1$, is as in the real experiment when $b=1$ and the view of $\mathcal{A}$, when $j=n-t$ and $\tilde{b}=0$, is as in the real experiment when $b=0$. Note that $\tilde{b}$ and j are the values randomly chosen by $\mathcal{B}$.

G and pk_D are set to be X_1,X_2 respectively and so are distributed correctly. All E_i for $i\in[n-t]$ are chosen identically to the experiment,except for E_j which is X_3 a random element of $\mathbb{G}$ and so distributed correctly. We can simulate the proof Ω_j due to the zero knowledge property of the proof of discrete logarithms. When computing the encrypted shares, although the secret key sk_D is not known to $\mathcal{B}$, we can instead use sk_i such that $\mathsf{sk}_i\cdot G=E_i$ for all $i\in[n]$. We have that $\mathsf{sk}_D\cdot E_i=\mathsf{sk}_i\cdot\mathsf{pk}_D$. In the case of corrupted parties, although we do not know these sk_i values, we can extract them from the proofs of knowledge Ω_i. In the case of the jth honest party, although again we do not know sk_j, as a DDH tuple is input, then $X_4=\mathsf{sk}_D\cdot E_j$ where $\mathsf{sk}_D\cdot G=\mathsf{pk}_D$. The proof $\mathsf{Pf}_{\mathsf{Sh}}$ can be simulated without knowledge of sk_D, due to the zero knowledge property. The oracle DIST can be simulated without knowledge of sk_D in the same way. The sk_i values that were extracted from the public keys E_i can be used to generate $\{C_i':i\in[n]\}$. The proof $\mathsf{Pf}_{\mathsf{Sh}}$ can again be simulated.

For all corrupted parties, it does not matter whether the polynomial f or f' is used to generate their encrypted share, because they have the same outputs on input α_i where $i\in[n-t+1,n]$. When $j=1$, and $\tilde{b}=1$, the polynomial f' is used to generate all of the encrypted shares for the honest parties. Therefore, the adversary is input S_0 and a correctly distributed sharing for S_1 and so the view is identically distributed to when $b=1$. When $j=n-t$ and $\tilde{b}=0$, the polynomial f is used to generate all of the encrypted shares for the honest parties. The adversary is input S_0 and a correctly distributed sharing for S_0 and so the view to $\mathcal{A}$ is identically distributed to when $b=0$.

Let $W_{j,d}$ be respectively the event that $\mathcal{A}$ outputs d when j is chosen at the beginning and a DDH tuple is input to $\mathcal{B}$. Where ϵ is the advantage of $\mathcal{A}$ defined above which is non–negligible, we have that

$$|\Pr\left[W_{n-t,1}|\tilde{b}=0\right]-\Pr\left[W_{1,1}|\tilde{b}=1\right]|=\epsilon.$$

Note that for $j^*=1,\ldots,n-t-1$, the view of the adversary when $j=j^*+1$ and $\tilde{b}=1$ and the view of the adversary when $j=j^*$ and $\tilde{b}=0$ is identically distributed so $\Pr\left[W_{j^*+1,1}|\tilde{b}=1\right]=\Pr\left[W_{j^*,1}|\tilde{b}=0)\right]$. Then

$$|\Pr\left[W_{n-t,1}|\tilde{b}=0\right]-\Pr\left[W_{1,1}|\tilde{b}=1\right]|=$$

$$\left|\sum_{j=1}^{n-t}\left(\Pr\left[W_{j,1}|\tilde{b}=0\right]-\Pr\left[W_{j,1}|\tilde{b}=1\right]\right)\right|.$$

Algorithm 42 $\mathcal{B}$

procedure $\mathsf{DIST}((\mathcal{U}, S'))$

  if $\mathcal{U} \not\subseteq [n+1,k]$ or $|\mathcal{U}| \neq n$ **then return** $\perp$
  end if
  Let $\{(\alpha_i, v_i) : i \in \mathcal{U}\}$ be the set $\{(\alpha_i, v_i) : i \in [n]\}$
  $(\{A'_i\}_{i\in[n]}, m'(X)) \leftarrow \mathsf{GShamir.Share}((\mathbb{G}, G, p, t, n, \{\alpha_i : i \in \mathcal{U}\}), S')$
  $\forall i \in \mathcal{U} \quad C'_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + A'_i$
  $f^* \leftarrow H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C'_i) : i \in \mathcal{U}\})$
  $V \leftarrow \sum_{i\in\mathcal{U}} v_i f^*(\alpha_i) \cdot C'_i$
  $U \leftarrow \sum_{i\in\mathcal{U}} v_i f^*(\alpha_i) \cdot E_i$
  Simulate proof $\mathsf{Pf_{Sh}}$ for G, pk_D, U, V
  return $(\{C'_i : i \in [n]\}, \mathsf{Pf_{Sh}})$
end procedure

procedure $\mathcal{B}(\mathbb{G}, p, X_1, X_2, X_3, X_4)$
  $\tilde{b} \leftarrow\$ \{0,1\}, j \leftarrow\$ [1, n-t]$
  $G \leftarrow X_1, x_0, x_1 \leftarrow\$ \mathbb{Z}_p, S_0 \leftarrow x_0 \cdot G, S_1 \leftarrow x_1 \cdot G$
  Choose pairwise distinct $\alpha_1 \in \mathbb{Z}_p, \cdots \alpha_n \in \mathbb{Z}_p$
  $\forall i \in [n] \quad v_i \leftarrow \prod_{j\in[n]\setminus\{i\}} (\alpha_i - \alpha_j)^{-1}$
  Randomly sample degree t polynomials $f, f' \in \mathbb{Z}_p[X]$ with $f(0) = x_0$, $f'(0) = x_1$, and $f(\alpha_i) = f'(\alpha_i)$ for $i \in [n-t+1, n]$
  $pp \leftarrow (\mathbb{G}, G, p, \{(\alpha_i, v_i) : i \in [n]\}); \mathsf{pk}_D \leftarrow X_2$
  $E_j \leftarrow X_3$; simulate the proof Ω_j for G, E_j, j; $\mathsf{pk}_j \leftarrow (E_j, \Omega_j)$
  $\forall i \in [n-t]\setminus\{j\} \quad \mathsf{sk}_i \leftarrow\$ \mathbb{Z}_p, E_i \leftarrow \mathsf{sk}_i \cdot G, \Omega_i \leftarrow \mathsf{DL}(\mathsf{sk}_i; G, E_i, i); \mathsf{pk}_i \leftarrow (E_i, \Omega_i)$
  $((\{\mathsf{pk}_i = (E_i, \Omega_i) : i \in [n-t+1, k]\}) \leftarrow \mathcal{A}(pp, \mathsf{pk}_D, \{\mathsf{pk}_i : i \in [n-t]\})$
  $\forall i \in [n-t+1, k]$ extract sk_i from Ω_i
  $\forall i \in [1, j-1] \quad C_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + f(\alpha_i) \cdot G$
  $\forall i \in [j+1, n-t] \quad C_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + f'(\alpha_i) \cdot G$
  $\forall i \in [n-t+1, n] \quad C_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + f(\alpha_i) \cdot G$

  if $\tilde{b} = 0$ **then** $C_j \leftarrow X_4 + f(\alpha_j) \cdot G$
  end if

  if $\tilde{b} = 1$ **then** $C_j \leftarrow X_4 + f'(\alpha_j) \cdot G$
  end if
  $f^* \leftarrow H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
  $V \leftarrow \sum_{i=1}^n v_i f^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^n v_i f^*(\alpha_i) \cdot E_i$
  Simulate proof $\mathsf{Pf_{Sh}}$ for G, pk_D, U, V
  $b' \leftarrow \mathcal{A}^{\mathsf{DIST}}(S_0, \{C_i : i \in [n]\}, \mathsf{Pf_{Sh}})$

  if $b' = \tilde{b}$ **then return** 1
  else return 0
  end if
end procedure

When a DDH tuple is input to $\mathcal{B}$, the probability $\mathcal{B}$ outputs 1 is

$$\frac{\sum_{j=1}^{n-t} 1/2 \Pr\left[W_{j,1}|\tilde{b}=1\right] + 1/2(1-\Pr\left[W_{j,1}|\tilde{b}=0\right])}{n-t}$$

$$= 1/2 + \frac{\sum_{j=1}^{n-t} \Pr\left[W_{j,1}|\tilde{b}=1\right] - \Pr\left[W_{j,1}|\tilde{b}=0\right])}{2(n-t)}.$$

Now consider the probability that $\mathcal{B}$ outputs 1 when a random tuple was input to $\mathcal{B}$. Because X_4 is now a uniform and independent variable, all inputs to $\mathcal{B}$ are independant of $\tilde{b}$. Therefore, $\mathcal{B}$ outputs 1 with probability 1/2.

As $|\frac{\sum_{j=1}^{n-t} \Pr[W_{j,1}|\tilde{b}=1]-\Pr[W_{j,1}|\tilde{b}=0])}{2(n-t)}| = \frac{\epsilon}{2(n-t)}$, which is non-negligible, then $\mathcal{B}$ has a non-negligible advantage in distinguishing DDH tuples.

Lemma 5 (Verifiability). *Our construction* DHPVSS *for a publicly verifiable secret sharing scheme satisfies* verifiability *if the hash function H is a random oracle.*

Proof. Verifiability of Key Generation. Our construction clearly satisfies verifiability of key generation because if $\mathsf{VerifyKey}(pp, id, \mathsf{pk} = (E, \Omega)) = 1$ then Ω is a valid proof of knowledge of the discrete logarithm for E. Therefore, sk such that $E = \mathsf{sk} \cdot G$ can be extracted from Ω.

Verifiability of Distribution. Our construction satisfies verifiability of distribution because if $\mathsf{Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i = (E_i, \Omega_i), C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1$ then $\mathsf{Pf}_{\mathsf{Sh}}$ is a valid proof of knowledge of discrete logarithm equality for G, pk_D, U, V, for $V = \sum_{i=1}^n v_i f^*(\alpha_i) \cdot C_i$, $U = \sum_{i=1}^n v_i f^*(\alpha_i) \cdot E_i$ where $f^* = H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$, and $\forall i \in [n] \quad v_i = \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}$. Therefore, sk_D such that $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$ and $V = \mathsf{sk}_D \cdot U$ can be extracted from $\mathsf{Pf}_{\mathsf{Sh}}$. As $V = \mathsf{sk}_D \cdot U$, then $\sum_{i=1}^n v_i f^*(\alpha_i) \cdot (C_i - \mathsf{sk}_D \cdot E_i) = 0$. Let event Φ be that $(C_1 - \mathsf{sk}_D \cdot E_1), \cdots (C_n - \mathsf{sk}_D \cdot E_n) \neq p(\alpha_1) \cdot G, \cdots, p(\alpha_n) \cdot G$ for any polynomial p of degree $\leq t$. Say r queries were made to the random oracle by the adversary. For event Φ to have occurred, some $\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$ was submitted to the random oracle and the polynomial f^* of degree $\leq n-t-1$ was returned such that $\sum_{i=1}^n v_i f^*(\alpha_i) \cdot (C_i - \mathsf{sk}_D \cdot E_i) = 0$. As $E_1, \cdots E_n$ are included in the input to the hash function and sk_D is defined by the input to the hash function, the probability of this is at most r/p, due to Theorem 1. We will assume that Φ did not occur which occurs with probability $1 - r/p$. We can determine the coefficients $\gamma_0, \cdots, \gamma_t$ of polynomial p by Lagrange interpolation. Then, letting $S = \gamma_0 \cdot G$, $(\{C_i : i \in [n]\}, \cdot) = \mathsf{DHPVSS.Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S)$ where the algorithm is run with the polynomial p chosen in $\mathsf{GShamirShare}$. Therefore, clearly a correctly formed sk_D, S and randomness for Dist exists.

Verifiability of Decryption. Our construction clearly satisfies verifiability of decryption because if $\mathsf{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk} = (E, \Omega), C, A', \mathsf{Pf}_{\mathsf{Dec}}) = 1$ then $\mathsf{Pf}_{\mathsf{Dec}}$ is a valid proof of knowledge of discrete logarithm equality for $G, E, \mathsf{pk}_D, C - A'$. Therefore, sk such that $E = \mathsf{sk} \cdot G$ and $C - A' = \mathsf{sk} \cdot \mathsf{pk}_D$ can be extracted from $\mathsf{Pf}_{\mathsf{Dec}}$. Therefore $A' = C - \mathsf{sk} \cdot \mathsf{pk}_D$, and so $\mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C) = (A', \cdot)$ for any randomness input to this algorithm.

F Communication complexity of PVSS

First note that any $\Pi_{\mathsf{NI-Pre}}$ consists of an element in $\mathcal{W}$ and one in $\mathbb{Z}_p$.

Communication complexity of HEPVSS. The communication of the algorithm HEPVSS.Dist consists of n ciphertexts in $\mathfrak{C}$ (the encryptions if the shares) and a proof $\mathsf{Pf_{Sh}}$, which is a $\Pi_{\mathsf{NI-Pre}}$ proof where $\mathcal{W} = \mathbb{G} \times \mathbb{Z}_p[X]_{\leq t} \times \mathfrak{R}^n$. When El Gamal encryption is used as $\mathcal{E}$, since $\mathfrak{R} = \mathbb{Z}_p$, $\mathfrak{C} = \mathbb{G}^2$ this amounts to a total of $(n+t+2)$ elements of $\mathbb{Z}_p$ and $2n+1$ in $\mathbb{G}$ which is roughly [8] equivalent to a total of $(3n+t+3)\log p$ bits.

On the other hand HEPVSS.DecShare communicates a decrypted message in $\mathbb{G}$ and the proof $\mathsf{Pf_{Dec}}$ where $\mathcal{W} = \mathcal{SK}$. In the case where we use El Gamal, the latter is 1 element in $\mathbb{G}$[9] and a challenge in $\mathbb{Z}_p$. Hence the communication is roughly $3\log p$ bits.

Communication complexity of DHPVSS. DHPVSS.Dist has smaller ciphertexts (1 group element each) and a smaller proof $\mathsf{Pf_{Sh}}$ consisting only of 2 elements in $\mathbb{Z}_p$. Hence the communication is in total roughly $(n+2)\log p$ bits, which is $3 \sim 3.5\times$ less than HEPVSS.Dist (depending on t).

The communication of DHPVSS.DecShare is as in HEPVSS.DecShare, hence it communicates $3\log p$ bits.

Comparison with SCRAPE and ALBATROSS. In SCRAPE and ALBATROSS, the encrypted shares of a secret $S = m(\alpha_0)G$ are given by $C_i = m(\alpha_i)\mathsf{pk}_i$ (where again $\mathsf{pk}_i = \mathsf{sk}_i G$). SCRAPE requires the dealer to commit to $m(\alpha_i)$ in a common base H by publishing $M_i = m(\alpha_i)H$ (n additional group elements), so that the SCRAPE trick can be used on the M_i's. Moreover the dealer needs to post non-interactive $\mathsf{DLEQ}(m(\alpha_i), \mathsf{pk}_i, C_i, H, M_i)$ for all i, which amounts to $n+1$ new $\mathbb{Z}_p$-elements. In total this means $(3n+1)\log p$ bits for the whole distribution. Instead ALBATROSS uses a standard homomorphic preimage proof of knowledge of the $m(X)$ underlying C_i. That is the dealer posts $\Pi_{\mathsf{NI-Pre}}(m(X), \{C_i\}_{i=1,n}, f)$ with $f(m(X)) = m(\alpha_i) \cdot \mathsf{pk}_i$. This requires $t+2$ $\mathbb{Z}_p$-elements, and so the communication complexity of the distribution phase is of $(n+t+2)\log p$ bits. Therefore, our DHPVSS scheme is the most communication efficient of all these alternatives.

Communication Complexity of Resharing. Resharing a secret among a committee of $n_{\mathsf{r}+1}$ parties requires, per party that is resharing their share, $(3n_{\mathsf{r}+1} + t_{\mathsf{r}+1} + 3)\log p$ bits, i.e. the same communication as to execute HEPVSS.Dist among the same set of parties. This means that we need at least $(t_{\mathsf{r}} + 1)(3n_{\mathsf{r}+1} + t_{\mathsf{r}+1} + 3)\log p$ bits in order for $\mathcal{C}_{\mathsf{r}}$ to reshare a secret to $\mathcal{C}_{\mathsf{r}+1}$.

The same happens with DHPVSS: the communication complexity per party who is resharing is $(n_{\mathsf{r}+1} + 2)\log p$ bits, which is the same as for distributing a share in the first place. This means $\mathcal{C}_{\mathsf{r}}$ needs to communicate in total $t_{\mathsf{r}}(n_{\mathsf{r}+1} + 2)\log p$ bits to reshare a secret to $\mathcal{C}_{\mathsf{r}+1}$.

[8] In practice, describing an element of an elliptic-curve group of order p requires slightly more information

[9] While it is true that, in order to force linearity of decryption, we have artificially set $\mathsf{sk}^* = (1, \mathsf{sk})$, and hence the keys are technically in $\mathbb{G}^2$, it is very easy to see that one only needs to send information related to the second coordinate.