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

Ignacio Cascudo$^{1*[0000-0001-5520-5386]}$, Bernardo David^{2†}, Lydia Garms^{1,3‡}, and Anders Konring^{2§}

^{1} IMDEA Software Institute, Madrid, Spain. ignacio.cascudo@imdea.org
^{2} IT University of Copenhagen, Copenhagen, Denmark. bernardo@bmdavid.com, konr@itu.dk
^{3} Keyless Technologies Limited. lydia.garms@keyless.io

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 with respect to 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. In addition,

[*]Ignacio Cascudo was supported by the Spanish Government under the project SecuRing (ref. PID2019-110873RJ-I00/MCIN/AEI/10.13039/501100011033), by the Madrid Government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union, 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).

we introduce a new PVSS with proof of sharing consisting of just two field elements, which as far as we know is the first achieving this, and may be of independent interest. 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 [1,12]: 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 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. Recent work [17] improves the work of [12] by achieving guaranteed output delivery in a constant number of rounds without relying on trusted setup.

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 [1] or MPC [12] 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 [12] and realized [1,14] 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 [1,14,5] 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 [1] 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 [1] was subsequently solved in [14] 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 [14] 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 [1], the concrete complexity is still quite high.

Encryption to the Future: A different approach is taken in [5], 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 [5] 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 [5] 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 [1,14,5] is that they are not amenable to publicly verifiable secret (re)sharing. Both in YOSO proactive secret sharing [1] and YOSO MPC [12], 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 [22,11,21,2,20,16] that different techniques for proving that a vector of encrypted shares are valid shares of a given secret. Recently, the SCRAPE [6] and ALBATROSS [7] 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 [13]. 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, which we call HEPVSS. This general result sheds new light on the construction on efficient PVSS schemes.

New PVSS with Minimal Overhead: Moreover, we introduce a new PVSS construction named DHPVSS with *constant-size proof of sharing correctness* which, as far as we know, is the first PVSS to achieve this. More precisely, the PVSS communicates only the n encrypted shares (which are one group element each) and two field elements for the proof. This may be of independent interest for other applications, such as randomness beacons.

Efficient PVSS for Anonymous Committees based on ECW: We instantiate our PVSS constructions 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 the full version of this paper [8], 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 [19] 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 [18], 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[4], 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 that a relaxed version of this abstraction allows to capture other encryption schemes with homomorphic properties such as latticed-based assumptions or Paillier in future work. In our second scheme DHPVSS, we introduce the idea of providing the dealer with an additional key pair 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 novelty is that, while in SCRAPE this needed an additional discrete logarithm equality (DLEQ) proof *for each share*, our new scheme requires *a single DLEQ proof.* This reduces the sharing correctness proof to only two $\mathbb{Z}_p$-elements while each encrypted shares is still one group element.

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, where most of our statements are instances of a general structure where we want to prove knowledge of preimage of some element via a *vector-space homomorphism* f: that is, let $\mathbb{F}$ be a finite field, $\mathcal{W}$ and $\mathcal{X}$ be $\mathbb{F}$-vector spaces, and $f : \mathcal{W} \to \mathcal{X}$ be a vector space homomorphism. Let

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

The standard (Schnorr-like) Σ-protocol Π_{Pre} for R_{Pre} is in Figure 1. It is easy to see it is a zero knowledge proof of knowledge with soundness error $1/|\mathbb{F}|$.

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

Proof of knowledge of witness w for x with respect to the relation $R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}$.
Public parameters: Finite field $\mathbb{F}$, vector spaces $\mathcal{W}, \mathcal{X}$ over $\mathbb{F}$, vector space homomorphism $f : \mathcal{W} \to \mathcal{X}$, $x \in \mathcal{X}$.
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

A non-interactive zero-knowledge (NIZK) proof of knowledge in the random oracle model is obtained by applying the Fiat-Shamir transform (Figure 2).

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 $pp = (\mathbb{F}, \mathcal{W}, \mathcal{X}, \mathcal{H})$.

```
$\Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$:
  $r \leftarrow_{\$} \mathcal{W}$, $a \leftarrow f(r)$, $e \leftarrow \mathcal{H}(x, a)$, $z \leftarrow r + e \cdot w$, return $\pi \leftarrow (e, z)$
$\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp, x, f, \pi)$:
  Parse $\pi = (e, z)$ and return accept if and only if $z \in \mathcal{W}$ and $e = \mathcal{H}(x, f(z) - e \cdot x)$.
```

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

Cyclic Group Homomorphism Preimage, DL Knowledge and DLEQ Knowledge Proofs. Some useful examples of homomorphism-preimage relations R_{Pre} are given by discrete logarithm and discrete logarithm equality. Indeed, 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.[4] Let G be a generator of $\mathbb{G}$. Given $X \in \mathbb{G}$, 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 $X, Y \in \mathbb{G}$ the discrete logarithm equality proof $\mathsf{DLEQ}(w; G, X, H, Y)$ is 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 properties are attained by El Gamal encryption scheme.

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, 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 $m_1, m_2 \in \mathfrak{C}$, $\rho_1, \rho_2 \in \mathfrak{R}$,*

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

Remark 1. $\mathbb{Z}_p$-linear homomorphic encryption schemes have simple NIZK of plaintext (and randomness) knowledge, implied by Figure 2 by taking $\mathcal{W} = \mathfrak{P} \times \mathfrak{R}$, $\mathcal{X} = \mathfrak{C}$ and the proof $\Pi_{\mathsf{NI-Pre}}((m, \rho); c, \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}})$ for the relation $R_{\mathsf{Enc}} = \{((m, \rho), c) \in \mathcal{W} \times \mathcal{X} : c = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m; \rho)\}$.

Proofs of Decryption Correctness We also need proofs of decryption correctness which keep the secret key hidden, i.e. NIZK 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.

[4]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.

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. $\mathcal{E}$ 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 1 $\mathcal{E}.\mathsf{ProveDec}(\mathsf{sk}, (\mathsf{pk}, m, c))$

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

Algorithm 2 $\mathcal{E}.\mathsf{VerifyDec}(\mathsf{pk}, m, c, \pi)$

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

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.

Here 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 is 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.

2.4 The SCRAPE Test

In SCRAPE [6], a technique for checking correctness of Shamir sharing in publicly verifiable secret sharing was introduced. Letting aside the details on how the technique works there, we are interested in the following fact, which in turn comes from well known results in coding theory [5].

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

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, $0 \le t < n < p$ are integers, and $\alpha_0, \alpha_1, \dots, \alpha_n \in \mathbb{Z}_p$ are pairwise distinct.

```
GShamir.Share(pp, S), where pp as above, and S ∈ G:
    $m(X) \leftarrow_{\$} \{m(X) \in \mathbb{Z}_p[X]_{\le t} : m(\alpha_0) = 0\}$
    for $i \in [n]$, $A_i \leftarrow S + m(\alpha_i) \cdot G$
    return $(A_1, \dots, A_n)$
GShamir.Rec($pp, I, (A_i)_{i \in I}$), where $I \subseteq [n], |I| = t + 1$ and $(A_i)_{i \in I} \in \mathbb{G}^{t+1}$:
    return $S' \leftarrow \sum_{i \in I} \lambda_{i,I} A_i$, where, for $i \in I$, $\lambda_{i,I} := \prod_{j \in I, j \ne i} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j}$
```

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

Theorem 1 (SCRAPE dual-code test). *Let $1 \le t < n$ be integers. Let p be a prime number with $p \ge n$. Let $\alpha_1, \dots, \alpha_n$ 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*

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

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

$$(\sigma_1, \dots, \sigma_n) \in C \quad \Leftrightarrow \quad \sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot \sigma_i = 0, \quad \forall m^* \in \mathbb{Z}_p[X]_{\le n-t-2}.$$

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 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.* [3]).

2.6 Encryption to the Future

We use the model for Encryption to the Future (EtF) from [5], 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 [15], which is used to state the definitions of [5] and that captures properties of natural Proof-of-Stake (PoS) based protocols such as [10]. We present a summary of the framework in the full version of the paper [8] 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 [10], 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 [15], $\mathcal{F}_{\mathsf{INIT}}$ is executed by the environment (as defined in the full version of the paper [8]). 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 [15] 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 [5], 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 $\mathsf{P}_1, \dots, \mathsf{P}_n$. 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 [5] 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 [5] 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 $\text{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ described in Algorithm 3, where a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ execute an underlying blockchain protocol with an environment $\mathcal{Z}$ as described in the full version of the paper [8]. 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 3 $\text{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$

```
$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_r(\mathcal{A}, \mathcal{Z}, 1^\lambda)$                  $\triangleright$ $\mathcal{A}$ executes $\Gamma$ with $\mathcal{Z}$ until round $r$
$(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view}^r_{\mathcal{A}})$         $\triangleright$ $\mathcal{A}$ outputs challenge parameters
$b \leftarrow_{\$} \{0,1\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m_b)$
$\mathsf{st} \leftarrow \mathcal{A}(\mathsf{view}^r_{\mathcal{A}}, \mathsf{ct})$                          $\triangleright$ $\mathcal{A}$ receives challenge $\mathsf{ct}$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$      $\triangleright$ Execute from $\mathsf{view}^r$ until round $\tilde{r}$
$(\tilde{\mathbf{B}}, b') \leftarrow \mathcal{A}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}}, \mathsf{st})$
if $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ then                  $\triangleright$ $\tilde{\mathbf{B}}$ is a valid evolution of $\mathbf{B}$
    if $sk^{\mathcal{A}}_{L,j} \notin \mathcal{W}_{\tilde{\mathbf{B}},\mathsf{sl},\mathsf{P}}$ then                 $\triangleright$ $\mathcal{A}$ does not win role $\mathsf{P}$
        return $b \oplus b'$
    end if
end if
return $\hat{b} \leftarrow_{\$} \{0,1\}$
```

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* $\lambda \in \mathbb{N}$*:*

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

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 [5] 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 4 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 4 $\mathrm{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}}$

```
$\mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A},\mathcal{Z},1^\lambda)$                    ▷ $\mathcal{A}$ executes $\Gamma$ with $\mathcal{Z}$
$(\mathbf{B},\mathsf{sl},\mathsf{S},m',\sigma') \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}_{\mathcal{A}})$
if $(m' \in \mathcal{Q}_{\mathsf{AfP}}) \vee (\mathsf{sk}^{\mathcal{A}}_{L,j} \in \mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{S}})$ then     ▷ $\mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}$ won or queried illegal $m'$
    return 0
end if
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r,\tilde{r})}(\mathcal{A},\mathcal{Z},1^\lambda)$          ▷ Execute from $\mathsf{view}^r$ until round $\tilde{r}$
$\tilde{\mathbf{B}} \leftarrow \mathsf{GetRecords}(\mathsf{view}^{\tilde{r}}_i)$
if $\mathsf{evolved}(\mathbf{B},\tilde{\mathbf{B}}) = 1$ then
    if $\mathsf{Ver}(\mathbf{B},\mathsf{sl},\mathsf{S},\sigma',m') = 1$ then                 ▷ $\mathcal{A}$ 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 5 and 6 with more than negligible probablity in the security parameter.*

Scenario 0 ($b = 0$) *In this scenario (5) 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 (6) is identical to scenario 0 but instead of interacting with $\mathcal{O}_{\mathsf{AfP}}$, the adversary interacts with a simulator $\mathcal{S}$.*

Algorithm 5 $b = 0$

```
$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_r(\mathcal{A},\mathcal{Z},1^\lambda)$
$\mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}^r_{\mathcal{A}})$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r,\tilde{r})}(\mathcal{A},\mathcal{Z},1^\lambda)$
return $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}})$
```

Algorithm 6 $b = 1$

```
$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_r(\mathcal{A},\mathcal{Z},1^\lambda)$
$\mathcal{A}^{\mathcal{S}}(\mathsf{view}^r_{\mathcal{A}})$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r,\tilde{r})}(\mathcal{A},\mathcal{Z},1^\lambda)$
return $b' \leftarrow \mathcal{A}^{\mathcal{S}}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}})$
```

We let $\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ *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.* $\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}} = 1$*) iff* $b' = b$*. Finally, an AfP scheme* $\mathcal{U}$ *is called private in the context of the blockchain* Γ *and underlying lottery predicate* $\mathsf{lottery}$ *if the following holds for a negligible function* μ*.*

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

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 (we give a precise description in the full version) 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 $\mathcal{H} : \{0,1\}^* \to [n]$ 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 7 $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$

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

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

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 full version [8])

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) [9]. A SoK scheme is a pair of algorithms $(\mathsf{SoK.sign}, \mathsf{SoK.verify})$ and is defined in 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 full version [8])

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 3. The construction in Figure 5 has AfP privacy as in Definition 8.*

(See proof sketch in full version [8])

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 (see full version [8]) to prove membership in a committee without directly revealing our public key in the setup.

3.4 AfP with Reusable Setup

In the full version [8], 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 the full version [8] 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 [21,20,16,6]. 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.

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 will allow for more efficient constructions as we will see in Section 4.3.

Setup

- $\mathsf{Setup}(1^\lambda)$ outputs public parameters pp.
- $\mathsf{DKeyGen}(pp)$, performed by the dealer, outputs a key pair $(\mathsf{pk}_D, \mathsf{sk}_D)$.
- $\mathsf{KeyGen}(pp, id_i)$, performed by i-th share receiver, outputs a key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{VerifyKey}(pp, id, \mathsf{pk})$, performed by a public verifier, outputs 0/1 (as a verdict on whether pk is valid).

Distribution

- $\mathsf{Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S)$ performed by the dealer, and where $S \in \mathbb{G}$ is a secret, outputs encrypted shares $C_i : i \in [n]$ and a proof $\mathsf{Pf}_{\mathsf{Sh}}$ of sharing correctness.

Verification

- $\mathsf{Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$ performed by the public verifier outputs 0/1 (as a verdict on whether the sharing is valid).

Reconstruction

- $\mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}_i, \mathsf{sk}_i, C_i)$, performed by a share receiver, outputs a decrypted share A_i and a proof $\mathsf{Pf}_{\mathsf{Dec}i}$ of correct decryption.
- $\mathsf{VerifyDec}(pp, \mathsf{pk}_D, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i})$ outputs 0/1 (as a verdict on whether A_i is a valid decryption of C_i).
- $\mathsf{Rec}(pp, \{A_i : i \in \mathcal{T}\})$ for some $\mathcal{T} \subseteq [n]$ of size $t+1$ outputs a secret S. We will only apply this algorithm to inputs where $\mathcal{T}$ is of size $t+1$ and such that all A_i have passed the verification check.

We let $\mathcal{PK}_D$ and $\mathcal{PK}$ contain all key pairs output by $\mathsf{DKeyGen}$ and KeyGen respectively. For non–deterministic algorithms we sometimes explicitly reference the randomness r input. For example, $\mathsf{Dist}(pp, \mathsf{pk}_D, sk_D, \{\mathsf{pk}_i : i \in [n]\}, S; r)$. One of our constructions will not require pk_D, sk_D and consequently $\mathsf{DKeyGen}$. In that case we omit these arguments from the inputs to the other algorithms.

We require a PVSS to satisfy correctness, verifiability and IND1-secrecy. We give these definitions in the full version of this paper.

4.2 HEPVSS: Generic PVSS from $\mathbb{Z}_p$-LHE Scheme

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.

The construction is relatively straightforward: the dealer construct the (group) Shamir sharing of the secret, and encrypts the shares using the $\mathbb{Z}_p$-LHE scheme, resulting in cyphertexts C_i. The sharing correctness proof needs to assert, not only that each C_i is individually a correct encryption, but also that the underlying plaintext messages are evaluations of a polynomial of degree at most t. Here we use the fact that the set of polynomials of degree at most t is a vector space, and the map that sends a polynomial to its evaluation in some point is linear, so we can capture the above statement in terms of knowledge of preimage of a certain linear map. For the proofs of security (correctness, indistinguishability of secrets and verifiability) we refer to the full version.

4.3 DHPVSS: A PVSS with Constant-Size Sharing Correctness Proof

We now give an optimized construction of a PVSS with a proof of sharing correctness consisting of just two field elements. The PVSS scheme, which we call DHPVSS, has IND1-secrecy under the DDH assumption.

We explain the idea of the construction next: 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 $\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot a_i \stackrel{?}{=} 0$, or alternatively

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

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 $\mathbb{Z}_p[X]_{\leq n-t-2}$. If it is not true that all a_i are of the form $m(\alpha_i)$ for some polynomial $m(X) \in \mathbb{Z}_p[X]_{\leq t}$, then the check succeeds with probability at most $1/p$.

In [6], the encrypted shares were $C_i = a_i \cdot \mathsf{pk}_i$. Because these are in different bases the check above cannot be directly applied on the C_i, and then the strategy consisted on sending additional elements $a_i \cdot H$ (for some group generator H), proving that the underlying a_i's are the same, and carrying out the check on these $a_i \cdot H$. All this introduces overhead which is linear in n.

Instead, in DHPVSS, the dealer has a key-pair $(\mathsf{sk}_D, \mathsf{pk}_D)$, with $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$, and encrypts A_i as $C_i = A_i + \mathsf{sk}_D \cdot E_i$, where $E_i = \mathsf{sk}_i \cdot G$ is an ephemeral public key of the i-th party. Note that $\mathsf{sk}_D \cdot E_i$ can be seen as a shared Diffie-Hellman key between dealer and the i-th party or, alternatively, C_i can be seen as an El-Gamal encryption of A_i under E_i with randomness sk_D.

```
Algorithms for Public Verifiable Secret Sharing Scheme HEPVSS

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 $pp = (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\}, \mathcal{E})$
HEPVSS.KeyGen($pp, id$):
  return $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \mathcal{E}.\mathsf{Gen}(1^\lambda)$
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
  $\mathcal{W} \leftarrow \mathbb{G} \times \mathbb{Z}_p[X]_{\leq t} \times \mathfrak{R}^n$,   $\mathcal{X} \leftarrow \{0\} \times \mathfrak{C}^n$,   $pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
  $w \leftarrow (S, m(X), \rho_1, \ldots, \rho_n)$,   $x \leftarrow (0, C_1, \ldots, C_n)$
  Let $f$ given by
    $f(w) := (m(\alpha_0), \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_1}(S + m(\alpha_1) \cdot G; \rho_1), \ldots, \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_n}(S + m(\alpha_n) \cdot G; \rho_n))$
  $\mathsf{Pf}_{\mathsf{Sh}} \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp_\pi, x, f)$
  return $(\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$
HEPVSS.Verify($pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}$):
  return $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, x, f, \mathsf{Pf}_{\mathsf{Sh}})$, with $\mathcal{W}, \mathcal{X}, pp_\pi, x, f$ as in HEPVSS.Dist
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 $(A, \mathsf{Pf}_{\mathsf{Dec}})$
HEPVSS.VerifyDec($pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}i}$):
  return $\mathcal{E}.\mathsf{VerifyDec}(A_i, C_i, \mathsf{pk}_i, \mathsf{Pf}_{\mathsf{Dec}i})$
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

The advantage is that now $\sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot A_i \overset{?}{=} O$ is equivalent to

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

which is *one single* DLEQ proof $\mathsf{DLEQ}(\mathsf{sk}_D; G, \mathsf{pk}_D, U, V)$ for *publicly computable*

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

One detail is that, as opposed to the PVSS in [6] (where $m^*(X)$ was locally sampled by the verifier), the prover needs to know $m^*(X)$ so this is sampled via a random oracle. The algorithms can be found in Figure 7 and Figure 8.

```
Algorithms for PVSS scheme DHPVSS, Setup and Distribution

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 $pp = (\mathbb{G}, G, p, t, n, \alpha_0, \{(\alpha_i, v_i) : i \in [n]\})$
DHPVSS.DKeyGen($pp$):
    $\mathsf{sk}_D \leftarrow_\$ \mathbb{Z}_p^*, \mathsf{pk}_D \leftarrow \mathsf{sk}_D \cdot G$
    return $(\mathsf{pk}_D, \mathsf{sk}_D)$
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 $(\mathsf{pk}, \mathsf{sk})$
DHPVSS.VerifyKey($pp, id, \mathsf{pk}$):
    parse $\mathsf{pk}$ as $(E, \Omega)$
    return accept iff $\Omega$ is valid w.r.t $G, E, id$
DHPVSS.Dist($pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S$):
    parse $\mathsf{pk}_i$ as $(E_i, \Omega_i)$, $pp$ as $(\mathbb{G}, G, p, t, n, \alpha_0, \{(\alpha_i, v_i) : i \in [n]\})$
    $pp_{\mathsf{Sh}} \leftarrow (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\})$
    $(\{A_i\}_{i \in [n]}, m(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathsf{Sh}}, S)$
    $\forall i \in [n], C_i \leftarrow \mathsf{sk}_D \cdot E_i + A_i$
    $m^* \leftarrow \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$ (for a RO $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p[X]_{\leq n-t-2}$)
    $V \leftarrow \sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot E_i$
    $\mathsf{Pf}_{\mathsf{Sh}} \leftarrow \mathsf{DLEQ}(\mathsf{sk}_D; G, \mathsf{pk}_D, U, V)$
    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 the full version.

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. Secret distribution in SCRAPE [6] requires $(3n+1)\log p$ bits, which was reduced to $(n+t+2)\log p$ bits in ALBATROSS [7]. Therefore DHPVSS.Dist obtains an additive saving of $t \log p$ bits with respect to the best previous alternative. The communication of both DHPVSS.DecShare and HEPVSS.DecShare is $3\log p$ bits. The share decryption complexities in [6] and [7] are similar to ours. More details can be found in the full version of this paper.

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,

```
Algorithms for PVSS scheme DHPVSS, Verification and Reconstruction

DHPVSS.Verify($pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}$):
    parse $\mathsf{pk}_i$ as $(E_i, \Omega_i), pp$ as $(\mathbb{G}, G, p, t, n, \{(\alpha_i, v_i) : i \in [n]\})$
    $m^* \leftarrow \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
    $V \leftarrow \sum_{i=1}^n v_i m^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^n v_i m^*(\alpha_i) \cdot E_i$
    return accept iff $\mathsf{Pf}_{\mathsf{Sh}}$ is valid w.r.t $G, \mathsf{pk}_D, U, V$
DHPVSS.DecShare($pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C$):
    parse $\mathsf{pk}$ as $(E, \Omega)$
    $A' \leftarrow C - \mathsf{sk} \cdot \mathsf{pk}_D$
    $\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathsf{DLEQ}(\mathsf{sk}; G, E, \mathsf{pk}_D, C - A')$
    return $(A', \mathsf{Pf}_{\mathsf{Dec}})$
DHPVSS.VerifyDec($pp, \mathsf{pk}_D, \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i}$):
    parse $\mathsf{pk}_i$ as $(E_i, \Omega_i)$
    return accept iff $\mathsf{Pf}_{\mathsf{Dec}i}$ is valid w.r.t $G, E_i, \mathsf{pk}_D, C_i - A_i$
DHPVSS.Rec($pp, \{A_i : i \in \mathcal{T}\}$):
    return GShamir.Rec($pp, \{A_i : i \in \mathcal{T}\}$)
```

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

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 $t_{\mathsf{r}+1}$. By design, the protocols will keep the secret hidden from any adversary corrupting at most t_{r} parties from $\mathcal{C}_{\mathsf{r}}$ and $t_{\mathsf{r}+1}$ 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 by 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 were for secrets over $\mathbb{Z}_p$. Each party in $\mathcal{C}_{\mathsf{r}}$ would hold $\sigma_\ell = m_{\mathsf{r}}(\alpha_\ell)$ where m_{r} is the sharing polynomial for that round, of degree t_{r}. A subcommittee L_{r} of $t_{\mathsf{r}} + 1$ parties in $\mathcal{C}_{\mathsf{r}}$ can then reshare the secret by PVSSing their shares among $\mathcal{C}_{\mathsf{r}+1}$ with Shamir scheme of degree $t_{\mathsf{r}+1}$. The parties in $\mathcal{C}_{\mathsf{r}+1}$ then compute the sum of the received shares weighted by coefficients $\lambda_{\ell,L_{\mathsf{r}}} := \prod_{j\in L_{\mathsf{r}}, j\neq \ell} \frac{\alpha_0 - \alpha_j}{\alpha_\ell - \alpha_j}$. Indeed, if we denote $[\sigma_\ell]$ the vector of shares sent by P_ℓ in L_{r}, then $\sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}}[\sigma_\ell] = \sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}}[m(\alpha_\ell)] = [\sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}} m(\alpha_\ell)] = [m(\alpha_0)]$.

In our situation, each party $\mathsf{P}_{\mathsf{r},i}$ in $\mathcal{C}_{\mathsf{r}}$ has instead a group element as share, and needs to PVSS it among $\mathcal{C}_{\mathsf{r}+1}$ using the algorithm Dist from previous section. However, the proof in Dist only guarantees that the distributed shares are consistent with some secret. Here we require in addition that this secret is the shared that the party has received previously.

To be more precise, in round r, each party $\mathsf{P}_{\mathsf{r},i}$ in committee $\mathcal{C}_{\mathsf{r}}$ has $A_{\mathsf{r},i}$ as share and in addition the encryption $C_{\mathsf{r},i} = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{r},i}}(A_{\mathsf{r},i})$ of $A_{\mathsf{r},i}$ is public. $\mathsf{P}_{\mathsf{r},i}$ now needs to create shares of $A_{\mathsf{r},i}$ for the committee $\mathcal{C}_{\mathsf{r}+1}$. Let $A_{i\to j}$ be the share that will be sent to $\mathsf{P}_{\mathsf{r}+1,j}$. This will be encrypted as $C_{i\to j} = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{r}+1,j}}(A_{i\to j})$

and $\mathsf{P}_{\mathsf{r},i}$ must prove that $C_{i\to j}$ are encryptions of a correct sharing whose secret is indeed the plaintext of $C_{\mathsf{r},i}$.

When a subset L_r of $\mathcal{C}_\mathsf{r}$ of $t_\mathsf{r}+1$ parties have correctly reshared, each $\mathsf{P}_{\mathsf{r}+1,j}$ sets $A_{\mathsf{r}+1,j} = \sum_{\ell\in L_\mathsf{r}} \lambda_{\ell,L_\mathsf{r}} A_{\ell\to j}$ as their share and the corresponding public ciphertext $C_{\mathsf{r}+1,j} = \sum_{\ell\in L_\mathsf{r}} \lambda_{\ell,L_\mathsf{r}} C_{\ell\to j}$ can be locally computed by everyone.

5.1 Resharing for HEPVSS

In the case of HEPVSS, the additional proof that the reshared value is the one corresponding to the public ciphertext can be integrated easily in HEPVSS.Dist if the encryption scheme has $\mathbb{Z}_p$-linear decryption. We give the construction and more details in the full version [8].

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 a key from the dealer. Here there are different dealers, i.e. the final share of each party in $\mathcal{C}_{\mathsf{r}+1}$ is a linear combination of shares sent by the parties in L_r. Thanks to the fact that the encryption is also a linear operation with respect to the public key of the sender, we can define a public key for committee L_r. Indeed, if we call pk_{D_ℓ} the public key of $P_{\mathsf{r},\ell}$ when acting as sender, then $\mathsf{pk}_{D,L_\mathsf{r}} := \sum_{\ell\in L_\mathsf{r}} \lambda_{\ell,L_\mathsf{r}} \cdot \mathsf{pk}_{D_\ell}$. Then we want to make sure that the output encryption for $P_{\mathsf{r}+1,j}$ is $C_{\mathsf{r}+1,j} = \mathsf{sk}_{\mathsf{r}+1,j} \cdot \mathsf{pk}_{D,L_\mathsf{r}} + \sum_{\ell\in L_\mathsf{r}} \lambda_{\ell,L_\mathsf{r}} A_{\ell\to j}$.

At the beginning of the resharing, each party $P_{\mathsf{r},i}$ in committee $\mathcal{C}_\mathsf{r}$ has as share $A_{\mathsf{r},i} = C_{\mathsf{r},i} - \mathsf{sk}_i \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$ where sk_i is the secret key for decrypting shares, and needs to create shares $A_{i\to j}$ of $A_{\mathsf{r},i}$ 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} (i.e. this party will create $C_{[n_{\mathsf{r}+1}]} = \{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}$ with $C_{i\to j} = \mathsf{sk}_{D_i} \cdot \mathsf{pk}_j + A_{i\to j}$) and prove their validity. In conclusion we need a proof for the following relation

$$\begin{aligned} R_{\mathsf{DHPVSS,Reshare}} =\{&(m(X), \mathsf{sk}_i, \mathsf{sk}_{D_i}); (pp, \mathsf{pk}_i, \mathsf{pk}_{D_i}, \mathsf{pk}_{D,L_{\mathsf{r}-1}}, \mathsf{pk}_{[n_{\mathsf{r}+1}]}, C_{\mathsf{r},i}, C_{[n_{\mathsf{r}+1}]}) : \\ &\mathsf{pk}_i = \mathsf{sk}_i \cdot G,\ \mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G,\ m(X) \in \mathbb{Z}_p[X]_{\le t},\ m(\beta_0) = 0, \\ &\text{and } \forall j \in [n_{\mathsf{r}+1}],\ C_{i\to j} = \mathsf{sk}_{D_i} \cdot \mathsf{pk}_j + A_{i\to j}, \\ &\text{where } A_{i\to j} = (C_{\mathsf{r},i} - \mathsf{sk}_i \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}) + m(\beta_j) \cdot G\} \end{aligned}$$

However, we also want to use the SCRAPE technique to reduce the size of the witness and hence of the proof. Note that if we set $U_j = C_{i\to j} - \mathsf{sk}_{D_i} \cdot \mathsf{pk}_j - C_{\mathsf{r},i} + \mathsf{sk}_i \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$ for all $j \in [n_{\mathsf{r}+1}]$ and $U_0 = O$, we want to make sure that for all $j \in [0, n_{\mathsf{r}+1}]$, $U_j = m(\beta_j) \cdot G$ for a polynomial of degree $\le t$ (in addition to the conditions $\mathsf{pk}_i = \mathsf{sk}_i \cdot G$ and $\mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G$).

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 $\sum_{j=0}^{n} v'_i \cdot m^*(\beta_j) \cdot U_j = O$, for a random polynomial m^* of degree $n-t-1$ (note here we apply Theorem 1 to a code of length $n+1$, rather than n).

Observe $\sum_{j=0}^{n} v'_j \cdot m^*(\beta_j) \cdot U_j = U' - \mathsf{sk}_{D_i} \cdot V' + \mathsf{sk}_i \cdot W'$ for publicly computable

$$U' := \sum_{j=1}^{n} v'_j \cdot m^*(\beta_j) \cdot (C_{i\to j} - C_{\mathsf{r},i}), \quad V' := \sum_{j=1}^{n} v'_j \cdot m^*(\beta_j) \cdot \mathsf{pk}_j, \text{ and}$$
$$W' := \sum_{j=1}^{n} v'_j \cdot m^*(\beta_j) \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}},$$

and therefore $P_{\mathsf{r},i}$ needs a proof of knowledge for

$$R'_{\mathsf{DHPVSS,Reshare},m^*} = \{(\mathsf{sk}_i, \mathsf{sk}_{D_i}); (\mathsf{pk}_i, \mathsf{pk}_{D_i}, U', V', W') : \\ \mathsf{pk}_i = \mathsf{sk}_i \cdot G, \ \ \mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G, \ \ U' = \mathsf{sk}_{D_i} \cdot V' - \mathsf{sk}_i \cdot W'\}$$

where we remark that now the witness only contains two elements but on the other hand relation depends on a polynomial $m^*(X)$ that has been sampled uniformly at random among polynomials of degree at most $n-t-1$. This leads to the protocol for PVSS resharing in Figure 9.

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 [1,12].

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.

Protocol for DHPVSS resharing

Participants: $\mathcal{C}_{\mathsf{r}} = \{P_{\mathsf{r},1}, \ldots, P_{\mathsf{r},n_{\mathsf{r}}}\}$ 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. "Sender" key pairs $(\mathsf{sk}_{D_i}, \mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G)$ for every party $\mathsf{P}_{\mathsf{r},i} \in \mathcal{C}_{\mathsf{r}}$, a "sender committee" public key $\mathsf{pk}_{D,L_{\mathsf{r}-1}}$, and "receiver" key pairs $(\mathsf{sk}_{r,i}, \mathsf{pk}_{r,i} = \mathsf{sk}_{r,i} \cdot G)$ for $\mathsf{P}_{r,i}$, where $r = \mathsf{r}, \mathsf{r}+1$, and $1 \leq i \leq n_r$; 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 $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p[X]_{\leq n-t-1}$, $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$. Let $\mathcal{W} \leftarrow \mathbb{Z}_p^2$, $\mathcal{X} \leftarrow \mathbb{G}^3$, and $pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H}')$.

Input: Public ciphertexts $C_{\mathsf{r},i} = \mathsf{sk}_{\mathsf{r},i} \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}} + A_{\mathsf{r},i}$ such that $A_{\mathsf{r},i} = h_{\mathsf{r}}(\alpha_i) \cdot G$ for some polynomial h_{r} of degree $\leq t_{\mathsf{r}}$.

Output: A public key $\mathsf{pk}_{D,L_{\mathsf{r}}}$ for a subset L_{r} of $\mathcal{C}_{\mathsf{r}}$, of size $t_{\mathsf{r}} + 1$. Public output ciphertexts $(C_{\mathsf{r}+1,1}, \ldots, C_{\mathsf{r}+1,n_{\mathsf{r}+1}})$ 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 $pp_{\mathrm{Sh},\mathsf{r}+1} = (\mathbb{G}, G, p, t_{\mathsf{r}+1}, n_{\mathsf{r}+1}, \{\beta_j : j \in [0, n_{\mathsf{r}+1}]\})$.
2. Resharing: For $i = 1, ..., n_{\mathsf{r}}$, $P_{\mathsf{r},i}$ does the following:
   (a) $A_{\mathsf{r},i} \leftarrow C_{\mathsf{r},i} - \mathsf{sk}_{\mathsf{r},i} \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$.
   (b) $(\{A_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, m_i(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathrm{Sh},\mathsf{r}+1}, A_{\mathsf{r},i})$.
   (c) For $j \in [n_{\mathsf{r}+1}]$, $C_{i\to j} \leftarrow \mathsf{sk}_{D_i} \cdot \mathsf{pk}_{\mathsf{r}+1,j} + A_{i\to j}$.
   (d) $m_i^*(X) \leftarrow \mathcal{H}(\{C_{\mathsf{r},i} : i \in [n_{\mathsf{r}}]\}, \mathsf{pk}_{D,L_{\mathsf{r}-1}})$.
   (e) $U_i' \leftarrow \sum_{j=1}^n v_j' \cdot m_i^*(\beta_j) \cdot (C_{i\to j} - C_{\mathsf{r},i})$, $V_i' \leftarrow \sum_{j=1}^n v_j' \cdot m_i^*(\beta_j) \cdot \mathsf{pk}_{\mathsf{r}+1,j}$, $W_i' \leftarrow (\sum_{j=1}^n v_j' \cdot m_i^*(\beta_j)) \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$.
   (f) $\pi_{\mathsf{r},i} \leftarrow \mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Prove}((\mathsf{sk}_{\mathsf{r},i}, \mathsf{sk}_{D_i}); pp_\pi, (\mathsf{pk}_{\mathsf{r},i}, \mathsf{pk}_{D_i}, U_i'), f_i)$, where $f_i(\mathsf{sk}_{\mathsf{r},i}, \mathsf{sk}_{D_i}) := (\mathsf{sk}_{\mathsf{r},i} \cdot G, \mathsf{sk}_{D_i} \cdot G, \mathsf{sk}_{D_i} \cdot V_i' - \mathsf{sk}_{\mathsf{r},i} \cdot W_i')$.
   (g) Output $\{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, \pi_{\mathsf{r},i}$.
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' and f_i as above (from public information and $P_{\mathsf{r},i}$'s output $\{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}$).
      ii. Compute $\mathit{\Pi}_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, (\mathsf{pk}_{\mathsf{r},i}, \mathsf{pk}_{D_i}, U_i'), f_i, \pi_{\mathsf{r},i})$.
   (b) Define L_{r} the set of $t + 1$ first indices for which the above proofs accept.
   (c) For $j \in [n_{\mathsf{r}+1}]$, $C_{\mathsf{r}+1,j} \leftarrow \sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L} \cdot C_{\ell\to j}$.
   (d) $\mathsf{pk}_{D,L_{\mathsf{r}}} \leftarrow \sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}} \cdot \mathsf{pk}_{D_\ell}$.
   (e) Output $(\{C_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, (\pi_{\mathsf{r},\ell})_{\ell \in L_{\mathsf{r}}}, \mathsf{pk}_{D,L_{\mathsf{r}}})$.

Fig. 9. Protocol for DHPVSS resharing

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. We make the following modifications to the HEPVSS algorithms in Figure 6, :

- **Communication:** All messages are posted to the underlying blockchain ledger used by the ECW scheme $\mathcal{E}$.
- $\mathsf{HEPVSS.Setup}(1^\lambda, t, n)$: 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 for HEPVSS (Section 5.1) to work with ECW. We show how to obtain an ECW based (and thus anonymous) resharing protocol in the full version of the paper [8].

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 our ECW is also compatible with these optimizations in the full version of the paper [8].

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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`.
6. 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.
7. 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.

8. Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. Cryptology ePrint, Report 2022/242, 2022. `https://eprint.iacr.org/2022/242`.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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`.
14. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 32–61. Springer, Heidelberg, November 2021.
15. 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.
16. 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.
17. Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Towards efficient YOSO MPC without setup. Cryptology ePrint Archive, Report 2022/187, 2022. `https://eprint.iacr.org/2022/187`.
18. 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.
19. 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.
20. A. Ruiz and J. L. Villar. Publicly verifiable secret sharing from Paillier's cryptosystem. In *Western European Workshop on Research in Cryptology 2005*, 2005.
21. 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.
22. Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.