Efficient and Universally Composable Single Secret Leader Election from Pairings

Dario Catalano[1], Dario Fiore[2], and Emanuele Giunta[2,3,4]

[1] Università di Catania, Italy.
catalano@dmi.unict.it
[2] IMDEA Software Institute, Madrid, Spain.
{dario.fiore,emanuele.giunta}@imdea.org
[3] Universidad Politecnica de Madrid, Spain.
[4] Scuola Superiore di Catania, Italy.

Abstract. Single Secret Leader Election (SSLE) protocols allow a set of users to elect a leader among them so that the identity of the winner remains secret until she decides to reveal herself. This notion was formalized and implemented in a recent result by Boneh, *et al.* (ACM Advances on Financial Technology 2020) and finds important applications in the area of Proof of Stake blockchains.

In this paper we put forward new SSLE solutions that advance the state of the art both from a theoretical and a practical front. On the theoretical side we propose a new definition of SSLE in the universal composability framework. We believe this to be the right way to model security in highly concurrent contexts such as those of many blockchain related applications. Next, we propose a UC-realization of SSLE from public key encryption with keyword search (PEKS) and based on the ability of distributing the PEKS key generation and encryption algorithms. Finally, we give a concrete PEKS scheme with efficient distributed algorithms for key generation and encryption and that allows us to efficiently instantiate our abstract SSLE construction.

Our resulting SSLE protocol is very efficient, does not require participants to store any state information besides their secret keys and guarantees so called *on-chain efficiency*: the information to verify an election in the new block should be of size at most logarithmic in the number of participants. To the best of our knowledge, this is the first efficient SSLE scheme achieving this property.

1 Introduction

Leader Election protocols are of fundamental importance to realize consensus in distributed systems. The rise of blockchain and its numerous applications brought renewed interest on this topic and motivated the need to consider consensus protocols that also provide some secrecy guarantees. This is the case, for example, of leader elections in the context of Proof of Stake blockchains (e.g., [AMM18, GHM$^+$17, KKKZ19, GOT19]) where one may wish to randomly select a *secret* leader, i.e., a leader that remains hidden until she reveals herself. In these contexts, leader-secrecy allows to protect against several attacks that would otherwise compromise the liveness of the blockchain. Indeed, if a malicious party could know the identity of a future leader, he could try to deny the leader's access to the network (using a denial of service attack, for instance) before the latter publishes her block, and this would affect, at least temporarily, the liveness and finality of the system. Bribery attacks could also be carried out with ease in order to influence the set of transactions that are going to be published.

Many existing solutions address this issue by secretly selecting a few potential leaders in *expectation* (e.g. [BGM16, BPS16]). This means that, for every given round, on expectation a single block leader is elected. Unfortunately, however, this also means that even many (or zero) leaders may be elected in any round.

This state of affairs led to the quest for an election protocol that secretly produces a *single* leader [Lab19], i.e., where *exactly* one single candidate is able to prove that she won the election. In principle this problem could be solved using general multiparty computation. What make such an approach problematic are however the efficiency requirements desired in a blockchain context. In particular, beyond being computationally efficient, the protocol should guarantee low communication complexity (i.e. the total number of exchanged messages should scale with $O(N)$ or better, where N is the number of miners/users), and more importantly it should be *on-chain efficient*: the amount of bits to store on chain, per new block, should be small (ideally logarithmic in N).

The question of finding such an election protocol was formally addressed in a recent work of Boneh *et al.* [BEHG20] who put forward the notion of *Single Secret Leader Election* (SSLE, from now on). Informally, an SSLE scheme is a distributed protocol that secretly elects a leader and satisfies *uniqueness* (at most one leader is elected), *fairness* (all participants have the same probability of becoming the leader) and *unpredictability* (if the adversary does not win the election, she should not be able to guess the leader better than at random). Boneh *et al.* [BEHG20] also proposed three constructions meeting this notion that are based on different approaches and that achieve different efficiency (and security) tradeoffs (cf. Table 1 for a summary).

Their first SSLE scheme relies on indistinguishability obfuscation (iO) [GGH$^+$13] and its main advantage is to achieve the lowest communication complexity and on-chain efficiency; indeed every election involves a single constant-size message from the winner. At the same time, given the status of iO realizations, this SSLE protocol is of very limited (if any) practical interest.

The second construction in [BEHG20] builds on Threshold Fully homomorphic Encryption (TFHE) [BGG$^+$18] and is asymptotically less efficient than the iO-based one: every election needs $O(t)$ communication (where t is a bound on the number of malicious users tolerated by the system) to partially decrypt a publicly computable ciphertext; after this round of communication, the winner can prove her victory. A nice aspect of the TFHE-based solution is that it actually requires only a *leveled* scheme for circuits that for, say, $N = 2^{16}$ participants, can be of depth as little as 10. However, other aspects of this solution make it far from practical. First, it is not on-chain efficient: to make the election verifiable, $O(t)$ bits of information must be stored in the new block (unless one applies a transformation through a general-purpose SNARK proof that t valid partial decryptions exist). Second, it requires large $O(N \log N)$ secret key shares, and no concrete distributed setup (for the TFHE scheme) is explicitly provided in [BGG$^+$18]. So to the best of our knowledge one would have to rely on general multiparty computation techniques to achieve it.

The third SSLE construction in [BEHG20] is based on shuffling and the decisional Diffie-Hellman assumption. Asymptotically, it performs worse than the other two solutions: every new election requires to communicate and store in the new block a freshly shuffled list of N Diffie-Hellman pairs[5] (along with a NIZK of shuffle). Notice that this makes the solution inherently not on-chain efficient. The authors also describe a lightweight variant whose communication costs are $O(\sqrt{N})$, but the tradeoff here is a scheme with significantly lower security guarantees, as the secret leader is selected in a public subset of only $\sqrt{N}$ users.

We note also that both the iO and TFHE-based SSLE protocols need a trusted setup. The latter must be realized with a distributed protocol and should be in principle refreshed when new users join the system. On the other hand, the shuffle-based solution is essentially setup-free and thus can handle more easily users that join and leave the system dynamically.

Beyond efficiency considerations, another fundamental limitation of the constructions above is that they are proved secure with respect to a (stand-alone) game-based definition which makes their actual security in concurrent settings unclear. This is problematic in practice as it is hardly the case that distributed consensus protocols are executed stand-alone.

Given this state of affairs, the main question that motivates our work is:
is it possible to build an SSLE protocol that is on-chain efficient and achieves good practical performances while also realizing strong composability guarantees?

1.1 Our contribution

In this paper we propose a new SSLE solution that answers the above question in the affirmative. Our first contribution is the proposal of a new definition of SSLE in the universal composability model [Can01] (see Section 3). We believe this to be the right notion to model security in the highly distributed, often concurrent, blockchain-like applications where electing a leader is required. Our new definition implies the game-based definition of Boneh *et al.* [BEHG20], but the converse is not true.

As a second contribution, we propose a UC-secure construction of SSLE. In particular, we give a generic protocol based on public key encryption with keyword search (PEKS) [BDOP04], and then propose an efficient instantiation of it based on pairings under the SXDH assumption. The latter is our main technical contribution: it is a protocol that achieves the same (asymptotic) communication complexity as the TFHE-based solution from [BEHG20] while achieving, in

[5] Precisely, when the winner no longer wants to participate in future elections, there is no need to shuffle for the next election; we ignore this special case in our analysis.

SSLE	**Security**	**Election efficiency**		
	model	Rounds	Comm.	On-chain
iO	Game-based	0	$O(1)$	$O(1)$
TFHE	Game-based	1	$O(t)$	$O(t)$
Shuffle-N	Game-based	1	$O(N)$	$O(N)$
Shuffle-$\sqrt{N}$	Game-based	1	$O(\sqrt{N})$	$O(\sqrt{N})$
Ours	UC	$1+1$	$O(t)$	$O(\kappa \log N)$

Table 1. Comparison between the SSLE solutions from [BEHG20] and the SSLE of this work. 'On-chain' refers to the amount of information to be stored on chain in the new block after every election. Shuffle-$\sqrt{N}$ achieves a weak unpredictability notion. Everywhere, in $O(\cdot)$ we include the fixed security parameter λ. κ is a statistical security parameter that gives meaningful security for $\kappa = \log N$.

addition, on-chain efficiency and much better practical performance. We refer to Table 1 for a comparison between ours and the previous solutions and to the next section for an overview of our protocol. We note that, although our protocol requires a total of 2 rounds of communication to prepare an election, the first round can actually be executed in a preprocessing phase and shared to prepare many elections, thus making the online rounds effectively 1, as in the other solutions. Moreover, the protocol *does not* require parties to keep any state across rounds of communication, besides their secret keys.

An overview of our SSLE protocol. Let us describe our protocol and its efficiency in slightly more detail. PEKS is a notion of functional encryption [BSW11, O'N10] in which given a ciphertext c encrypting a keyword w and secret key sk associated to another keyword w', the decryption allows one to learn if $w = w'$ and nothing more. Our SSLE protocol is based on the following simple idea. For every election a small subset of users generates a ciphertext c that encrypts a random keyword $j \in \{0, \dots, N-1\}$. At registration time, each user is given a secret key sk_i associated to an integer i, and can claim victory by giving a NIZK proof that she can decrypt the election's ciphertext.

More specifically, our protocol consists of two phases: (1) a setup (done rarely) in which the users run an MPC protocol to generate the public key of the PEKS and distribute its secret keys, (2) an election's procedure in which a randomly sampled committee of κ players generates a commitment to the election's ciphertext in a distributed way. The commitment is then opened in a distributed way. Whoever knows a secret key that decrypts the ciphertext is the leader.

We formalize this approach in a generic SSLE protocol that we prove UC-secure assuming ideal functionalities for the setup and encryption algorithms of any PEKS (see Section 4). Our main technical contribution, however is to design an efficient instantiation of this blueprint, by showing an "MPC-friendly" PEKS and by proposing very efficient (distributed) protocols for the setup and election phases. To devise such a PEKS we build on (a modified variant of) the functional encryption for orthogonality (OFE) scheme recently proposed by

Wee [Wee17]. Furthermore we extend this functionality to test keywords equality mod N albeit the message space is over a large field $\mathbb{F}_q$. We refer to this new primitive as *modular PEKS*.

Informally, the committed ciphertexts created in the election procedure are (plain) El Gamal encryptions of Wee's ciphertexts. An immediate advantage of this approach is that it allows for a very efficient setup procedure: it merely consists in a threshold key generation for El Gamal followed by the key generation for the functional encryption scheme. When relying on a publicly available random beacon, we show that the latter can be realized efficiently in two rounds of communication, one of which only used to perform complaints.

More interestingly, however, our proposed scheme allows to complete step (2) efficiently both in terms of computation and communication. Indeed, our protocol manages to distributively create valid (committed) ciphertexts c (encrypting messages uniformly distributed in a given range) in one single round of communication! Moreover, this round of communication can be used to generate, in parallel, as many committed ciphertexts as one wishes, one for every future election. This way, the communication needed to perform an election effectively consists of only one round of communication in which $O(t)$ parties send their partial opening of the election's ciphertext.

We note that the naïve approach of posting all these $O(t)$ partial openings in the blockchain would destroy our claimed on-chain efficiency guarantees. Interestingly, we can do better than this. Parties can exchange the $O(t)$ partial openings off-chain and store on-chain only much shorter aggregate values that still enable anyone to verify the correctness of the election. Recall that opening our committed ciphertexts consists in, distributively, decrypting corresponding El Gamal ciphertexts. Simplifying things a bit, in our case this is achieved by letting players exchange partial decryption shares $(K_{1,i}, K_{2,i})$ together with corresponding NIZKs. These shares are then (locally) multiplied together to get values (K_1, K_2) that can be used to retrieve the encrypted ciphertext c. Whoever is able to decrypt c correctly can then claim victory. Concretely, in our protocol, a user can claim victory by posting on the blockchain only (K_1, K_2), together with a proof that she can correctly decrypt c. Surprisingly, we show that a potentially expensive aggregated NIZK proving correctness of (K_1, K_2) is not needed for our protocol to be secure, as we prove that coming up with different (K_1', K_2') which open the ElGamal commitment to another $c' \neq c$ that an adversary is able to decrypt, implies being able to break the underlying functional encryption scheme.

Concrete efficiency and comparison to previous solutions. To confirm the concrete performances of our protocol we measure them for $N = 2^{14}$ users, as suggested in [Lab19]. Our results show that the communication costs of an election are 34.0 KB to generate the committed election's ciphertext, 1.57 MB for the partial decryptions, and 256 B to claim victory. Importantly, out of all this information, only 34.3 KB per election have to be stored on-chain for verifiability.

The major cost in our protocol is that of setup, which for 2^{14} users would amount to 252 MB. This setup, however, is supposed to be performed rarely[6]. Indeed, in our protocol we can add new users to the system without running a full setup: they engage in a registration procedure that allows them to receive their secret keys, without altering the key material of other users. This can be done with only 73 KB of communication per registration. If we compare to the shuffle-N solution of Boneh et al. [BEHG20], our protocol can easily amortize the expensive setup and results in less communication. In the shuffle-N solution, the issue is that every time a new user is added (which always includes the winner of the previous election if he still wants to run) a new shuffle has to be communicated and posted on-chain: this is about 1 MB per shuffle for 2^{14} users. Concretely, if we assume 50 new users join before every election,[7] after 100 elections the shuffle-N scheme generates 6.2 GB to be communicated and stored on-chain, whereas our protocol involves 1.8 GB of off-chain communication and only 5.9 MB of on-chain storage.

Our election protocol more in detail. At the heart of our protocol there is a very efficient method to generate committed ciphertexts of the form discussed above. Here we informally highlight the main ideas underlying this construction. Recall that we build our PEKS from a tailored variant of the functional encryption for orthogonality (OFE) scheme recently proposed by Wee [Wee17]. In OFE a ciphertext is associated to a vector $\mathbf{x}$, a secret key corresponds to a vector $\mathbf{y}$ and decryption allows one to learn if $\mathbf{y}^\top \mathbf{x} = 0$. The basic idea of our (modular) PEKS from OFE is inspired to a transformation from [KSW08] with a novel tweak.

In what follows, to keep the presentation intuitive, we present a simplified version of our methods that, in particular, supports vectors of dimension 2 (rather than 3 as in our actual scheme) and only allows to test equality of keywords (rather than equality mod N).

During setup, each party P_i receives a public and secret key $\mathsf{mpk}, \mathsf{sk}_i$ of the OFE scheme, where sk_i is associated to the vector $(1, i)$. If there were a magic way to directly produce an encryption c of $(m, -1)$ such that m is uniform over $[N]$ (and no user gains any extra information on m), then, using $\mathsf{FE.Dec}$, each party could test if $m = i$ by simply checking whether $(m, -1)^\top (1, i) = 0$. Clearly the only user able to do this could then claim victory. Unfortunately, since no such wizardry is currently known, we go for the next best option: we develop a very fast, one round protocol to jointly produce a *commitment* of such a c[8]. The commitment is just a (standard) El Gamal encryption of c that can be (distributively) opened in one round of communication.

[6] As in the TFHE solution, our protocol in practice requires periodic setup to refresh the secrets shared when many new users join (see Sec. 6 for a discussion on this).

[7] This number is justified by [Lab19], where $O(\log^2 N)$ new users are expected.

[8] We stress here that no efficient single round solution to directly produce c seems possible because of rushing attacks.

In this informal presentation, we explain how to generate the (committed) ciphertext, in the simpler case where m is allowed to lie in the slightly larger interval $[\kappa N]$. Our underlying ciphertexts have the following shape[9]

$$\mathbf{c}_0 = [s\mathbf{a}]_1, \quad c_1 = \left[m\sigma + s\mathbf{a}^\top \mathbf{w}_1\right]_1, \quad c_2 = \left[-\sigma + s\mathbf{a}^\top \mathbf{w}_2\right]_1.$$

where $[\mathbf{a}]_1, \left[\mathbf{a}^\top \mathbf{w}_1\right]_1, \left[\mathbf{a}^\top \mathbf{w}_2\right]_1$ are public key elements and s, σ are random values.[10] Using the random beacon, we begin by generating a (small) election committee $Q \subseteq [N]$ of size κ and two (random) group elements G, H that can be interpreted as an ElGamal encryption of $[\sigma]_1$ in the following way

$$G = g^\theta, \qquad H = h^\theta [\sigma]_1$$

where (g, h) is the El Gamal public key and θ, σ are random and unknown to participants. Using this public information, each player $P_i \in Q$ can create (committed) encryptions of m_i by simply choosing random r_i, ρ_i, s_i, and $m_i \in [N]$ and broadcasting $[s_i\mathbf{a}]_1$ together with

$$\begin{array}{ll} G^{m_i} \cdot g^{r_i} = g^{\theta m_i + r_i} & H^{m_i} \cdot h^{r_i} \cdot \left[s_i\mathbf{a}^\top \mathbf{w}_1\right]_1 = h^{\theta m_i + r_i} \left[m_i\sigma + s_i\mathbf{a}^\top \mathbf{w}_1\right]_1 \\ G^{-1} \cdot g^{\rho_i} = g^{\rho_i - \theta} & H^{-1} \cdot h^{\rho_i} \cdot \left[s_i\mathbf{a}^\top \mathbf{w}_2\right]_1 = h^{\rho_i - \theta} \left[-\sigma + s_i\mathbf{a}^\top \mathbf{w}_2\right]_1 \end{array}$$

All these (committed) ciphertexts share the same randomness σ and can thus be multiplied together to produce the final (committed) ciphertext of the vector $(m = \sum_{i \in Q} m_i, -1)$. Note that the message m lies in the larger interval $[\kappa N]$ but $m \bmod N$ is uniform over $[N]$ as long as so is at least one of the m_i's. Finally, as mentioned earlier, our actual realization (cf. Section 2.5) works around this issue by managing to test equalities modulo N.

1.2 Other related work

Recently, the importance of SSLE solutions was confirmed by a study by Azouvi and Cappelletti [AC21]. Their analysis shows substantial security gains (when compared to probabilistic election schemes) both when considering the private attack (the worst attack on longest-chain protocols [DKT+20]) and grinding attacks. The problem of extending proof of stake systems to consider privacy was considered, among others, in [GOT19] and in [KKKZ19]. Leader election protocols were also considered by Algorand [GHM+17] and Fantomette [AMM18]. There the idea is to first identify few potential leaders (via a VRF) that then reveal themselves in order and choose the winner via some simple tie break method (e.g. lowest VRF output wins). The approach is efficient but has the drawback that the elected leader does not know she was elected until everybody else published their value. Moreover, implicitly requires all nodes to be able to see the winner's output: users not getting this information might incorrectly think that another leader was elected (causing the chain to fork). We stress that this cannot happen in our setting.

[9] For clarity note that group operations are denoted multiplicatively, and that we make use of the bracket notation, cf. Section 2.1.

[10] In Wee's scheme $\sigma = s\mathbf{a}^\top \mathbf{u}$, with $\left[\mathbf{a}^\top \mathbf{u}\right]_1$ being an extra element of the public key.

1.3 Organization

In the next section we start by introducing notation, computational assumptions and cryptographic primitives used by our schemes. There we also recall the game-based definition of SSLE from [BEHG20]. Next, in section 3 we give our definition of SSLE in the universal composability framework. Section 5 includes our main contribution, that is our efficient SSLE protocol from the SXDH assumption (the generic SSLE construction from PEKS is given in section 4). Finally, in section 6 we discuss the efficiency of our protocol in a realistic scenario and compare it with the SSLE based on shuffles by Boneh et al. [BEHG20].

2 Preliminaries

2.1 Notation

$\lambda \in \mathbb{N}$ denotes the security parameter. A function $\varepsilon(\lambda)$ is said *negligible in* λ if it vanishes faster than the inverse of any polynomial in λ. $[n] = \{0, \dots, n-1\}$. Bold font $(\mathbf{a}, \mathbf{u}, \mathbf{w}, \dots)$ denotes vectors with entries in a given field or a group. $x \leftarrow^{\$} S$ means that x is sampled uniformly and with fresh randomness from S. N is the number of players and t the threshold parameter.

We denote with $\mathcal{G}(\lambda)$ a *bilinear group generator*, that is an algorithm which returns the description of a bilinear group $\mathsf{bg} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of the same prime order $q > 2^\lambda$, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are two generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear map. We use $g_T = e(g_1, g_2)$ as a canonical generator of $\mathbb{G}_T$. When $\mathbb{G}_1 = \mathbb{G}_2$, the groups are called *symmetric*; otherwise they are called *asymmetric*. In our work we use Type-III *asymmetric* bilinear groups [GPS08] where no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ is known.

$\mathbb{F}_q$ is the finite field of prime cardinality q. Given a vector $\mathbf{a} = (a_i)_{i=1}^n \in \mathbb{F}_q^n$ and a group element g we denote $[\mathbf{a}]_g = (g^{a_1}, \dots, g^{a_n})$. When the base is g_1, g_2 or g_T we replace the above notation with $[\mathbf{a}]_1$, $[\mathbf{a}]_2$ and $[\mathbf{a}]_T$ respectively. Operations with vectors in $\mathbb{G}^n$ are entry-wise, i.e., for $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$, $\mathbf{g} \cdot \mathbf{h} = (g_i \cdot h_i)_{i=1}^n$, $\mathbf{g}^a = (g_i^a)_{i=1}^n$. Pairings are the only exception where $e(\mathbf{g}, \mathbf{h}) = e(g_1, h_1) \cdot \ldots \cdot e(g_n, h_n)$ for $\mathbf{g} \in \mathbb{G}_1^n$ and $\mathbf{h} \in \mathbb{G}_2^n$. Similarly $\mathbf{g}^{\mathbf{a}} = g_1^{a_1} \cdot \ldots \cdot g_n^{a_n}$.

2.2 SXDH assumption

Our efficient construction relies on the SXDH assumption in bilinear groups, which informally states that the classical DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$. More formally,

Definition 1 (SXDH assumption). *Let $\mathcal{G}$ be a bilinear group generator. We say that the* SXDH *assumption holds for $\mathcal{G}$ if for every PPT adversary $\mathcal{A}$, and every $s \in \{1, 2\}$ there exists a negligible function ε such that:*

$$|\Pr[\mathcal{A}(\mathsf{bg}, [a]_s, [b]_s, [c]_s) = 1] - \Pr[\mathcal{A}(\mathsf{bg}, [a]_s, [b]_s, [ab]_s) = 1]| \leq \varepsilon(\lambda)$$

where the probabilities are over the random choice of $a, b, c \leftarrow^{\$} \mathbb{F}_q$ and $\mathsf{bg} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \leftarrow^{\$} \mathcal{G}(1^\lambda)$.

When the above assumption is considered in only one group $\mathbb{G}_s$, for either $s = 1$ or $s = 2$, we refer to it as DDH in $\mathbb{G}_s$. We call DDH^0 a game in which $\mathcal{A}$ received the first distribution and DDH^1 a game in which he receives the second one.

In the paper we also use an extension of DDH for vectors of n elements, called DDH_n, which says it is hard to distinguish $([a_1]_s, \ldots, [a_n]_s, [b]_s, [c_1]_s, \ldots, [c_n]_s)$, denoted as DDH_n^0, from $([a_1]_s, \ldots, [a_n]_s, [b]_s, [a_1 b]_s, \ldots, [a_n b]_s)$ denoted as DDH_n^1, for random $a_i, b, c_i \in \mathbb{F}_q$. We note that DDH_n can be reduced to DDH in the same group [NR97].

2.3 Functional Encryption

We recall the definition of Functional Encryption [BSW11, O'N10].

Definition 2. *A* functionality F *is a family of functions* $\mathsf{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$, *where* $\mathcal{X}$ *is the plaintext space and* $\mathcal{Y}$ *is the output space.*

Definition 3. *A* ***functional encryption scheme*** *for a functionality* F *is a tuple* $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$ *of* PPT *algorithms such that*

- $\mathsf{FE.Setup}(1^\lambda) \xrightarrow{\$} (\mathsf{mpk}, \mathsf{msk})$ *generates the secret and public master keys.*
- $\mathsf{FE.Enc}(m, \mathsf{mpk}; r) \to c$ *returns a ciphertext. Randomness* r *may be omitted.*
- $\mathsf{FE.KeyGen}(f, \mathsf{msk}) \xrightarrow{\$} \mathsf{sk}_f$ *returns a key associated to the function* $f \in \mathsf{F}$.
- $\mathsf{FE.Dec}(c, f, \mathsf{mpk}, \mathsf{sk}_f) \to x$ *a bit string.*

The scheme is ***correct*** *if for any* $m \in \mathcal{X}$ *and* $f \in \mathsf{F}$, *sampled* $\mathsf{mpk}, \mathsf{msk} \xleftarrow{\$} \mathsf{FE.Setup}(1^\lambda)$, $c \xleftarrow{\$} \mathsf{FE.Enc}(m, \mathsf{mpk})$, $\mathsf{sk}_f \xleftarrow{\$} \mathsf{FE.KeyGen}(f, \mathsf{msk})$, *then up to negligible probability* $\mathsf{FE.Dec}(c, f, \mathsf{mpk}, \mathsf{sk}_f) = f(m)$.

We recall the notion of selectively secure FE, which suffices for our goals.

Definition 4. *A functional encryption scheme achieves* ***selective security*** *if for any* PPT *algorithm* $\mathcal{A}$ *there exists a negligible function* ε *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{SSFE}}(1^\lambda) = \left| \Pr\left[\mathsf{Exp}^{\mathcal{A}}_{\mathsf{SSFE}}(1^\lambda) = 1\right] - \frac{1}{2} \right| \le \varepsilon(\lambda).$$

2.4 Functional Encryption for Modular Keyword Search

Recall that the keyword search functionality [BDOP04, ABC$^+$05] is defined as $\mathsf{F}_{\mathsf{ks}} = \{f_y : \mathcal{X} \to \{0, 1\}\}$, where each function $f_y \in \mathsf{F}_{\mathsf{ks}}$ labelled by $y \in \mathcal{X}$ is such that $f_y(x)$ returns 1 if $x = y$ and 0 otherwise. Our realization works with a generalisation of the above where equality are checked modulo a given integer. Formally we consider the modular keyword search functionality $\mathsf{F}^{\kappa}_{\mathsf{mks}} = \{f_y : \mathbb{F}_q \times \mathbb{F}_q \to \{0, 1\}\}$ parametrized by a positive integer κ of polynomial size, where each function f_y labelled by $y \in \mathbb{F}_q$ are such that $f_y(x, n)$ returns 1 if $x = y + \delta n$ for some $\delta \in [\kappa]$, and 0 otherwise. Observe that when $y \in [n]$ and $x \in [\kappa n]$, then $f_y(x, n) = 1$ if and only if $x = y \mod n$.

```
**Selective Security Game** $\mathsf{Exp}^{\mathcal{A}}_{\mathsf{SSFE}}(1^\lambda)$:
1 : $m_0, m_1 \leftarrow^{\$} \mathcal{A}$
2 : Sample $b \leftarrow^{\$} \{0,1\}$, $\mathsf{mpk}, \mathsf{msk} \leftarrow^{\$} \mathsf{FE.Setup}(1^\lambda)$, $c \leftarrow^{\$} \mathsf{FE.Enc}(m_b, \mathsf{mpk})$
3 : Send $\mathcal{A} \leftarrow \mathsf{mpk}, c$
4 : When $\mathcal{A}$ queries $f \in \mathsf{F}$, if $f(m_0) \neq f(m_1)$ then $\mathcal{A} \leftarrow \perp$. Otherwise:
5 :     Compute $\mathsf{sk}_f \leftarrow^{\$} \mathsf{FE.KeyGen}(f, \mathsf{msk})$ and send $\mathcal{A} \leftarrow \mathsf{sk}_f$
6 : When $\mathcal{A} \rightarrow b'$: Return $b == b'$
```

Fig. 1. Selective security game for a FE scheme with functionality F

2.5 Our Realization of FE for Modular Keyword Search

We realize our FE scheme for the keyword search functionality $\mathsf{F}^{\kappa}_{\mathsf{mks}}$ through a more powerful scheme for the so-called *orthogonality* functionality [KSW08]. In the latter we have the message space $\mathcal{X} = \mathbb{F}_q^n$ and each function $f_{\mathbf{y}}$, defined by a vector $\mathbf{y} \in \mathbb{F}_q^n$, is such that $f_{\mathbf{y}}(\mathbf{x})$ returns 1 when $\mathbf{y}^\top \mathbf{x} = 0$ and 0 otherwise.

A general construction of FE for F_{ks} from an OFE scheme already appears in previous work [KSW08]. In this paper, we tweak that template in order to support the $\mathsf{F}^{\kappa}_{\mathsf{mks}}$ described earlier (see Fig. 2). The idea is that $m = \gamma + \delta n$ if and only if $(m, -1, -n)^\top(1, \gamma, \delta)$ for some $\delta \in [\kappa]$. Therefore, using an OFE scheme with dimension 3, a ciphertext for m and n is an encryption of the vector $\mathbf{x}_{m,n} = (m, -1, -n)$, while a key for γ is a collection of keys for the vectors $\mathbf{y}_{\gamma,\delta} = (1, \gamma, \delta)$, with $\delta \in [\kappa]$. This way, decryption can be realized by testing if one of the keys successfully decrypts.

```
$\mathsf{MKS.Setup}(1^\lambda)$:
$(\mathsf{mpk}', \mathsf{msk}') \leftarrow^{\$} \mathsf{FE.Setup}(1^\lambda, 3)$
**Return** $\mathsf{mpk}', \mathsf{msk}'$

$\mathsf{MKS.KeyGen}(\gamma, \mathsf{msk})$:
**For** $\delta \in [\kappa]$:
    $\mathsf{sk}_{\gamma,\delta} \leftarrow^{\$} \mathsf{FE.KeyGen}((1, y, \delta), \mathsf{msk})$
**Return** $\mathsf{sk}_\gamma \leftarrow (\mathsf{sk}_{\gamma,0}, \ldots, \mathsf{sk}_{,\gamma,\kappa-1})$

$\mathsf{MKS.Enc}(m, n, \mathsf{mpk})$:
$\mathbf{x}_{m,n} \leftarrow (m, -1, -n)$
**Return** $c \leftarrow^{\$} \mathsf{FE.Enc}(\mathbf{x}_{m,n}, \mathsf{mpk})$

$\mathsf{MKS.Dec}(c, y, \mathsf{mpk}, \mathsf{sk}_y)$:
Set $\mathbf{y}_{\gamma,\delta} \leftarrow (1, \gamma, \delta)$ for all $\delta \in [\kappa]$
**If** $\exists \delta \in [\kappa]$ : $\mathsf{FE.Dec}(c, \mathbf{y}_{\gamma,\delta}, \mathsf{mpk}, \mathsf{sk}_{\gamma,\delta}) = 1$
    **Return** 1.   **Else Return** 0
```

Fig. 2. Our FE for $\mathsf{F}^{\kappa}_{\mathsf{mks}}$ from and orthogonality functional encryption scheme

Note however that the resulting construction is secure under the weaker notion in which the adversary, who initially queries an encryption of (m_0, n_0) and

```
$\mathsf{FE.Setup}(1^\lambda, n)$:
    Sample $\mathbf{a}, \mathbf{w}_1, \ldots, \mathbf{w}_n \leftarrow^{\$} \mathbb{F}_q^2$
    $\mathsf{mpk} \leftarrow \left([\mathbf{a}]_1, \left[\mathbf{a}^\top \mathbf{w}_1\right]_1, \ldots, \left[\mathbf{a}^\top \mathbf{w}_n\right]_1\right)$
    $\mathsf{msk} \leftarrow (\mathbf{w}_i)_{i=1}^n$ and return $(\mathsf{mpk}, \mathsf{msk})$

$\mathsf{FE.KeyGen}(\mathbf{y}, \mathsf{msk})$:
    $r \leftarrow^{\$} \mathbb{F}_q \setminus \{0\}$
    Return $\mathsf{sk}_{\mathbf{y}} \leftarrow \left[\sum_{i=1}^n r y_i \mathbf{w}_i\right]_2, [r]_2$

$\mathsf{FE.Dec}(c, \mathbf{y}, \mathsf{mpk}, \mathsf{sk}_{\mathbf{y}})$:
    Parse $c = (\mathbf{c}_0, c_i)_{i=1}^n$ with $\mathbf{c}_0 \in \mathbb{G}_1^2$
    Parse $\mathsf{sk}_{\mathbf{y}} = (\mathbf{d}_0, d_1)$ with $\mathbf{d}_0 \in \mathbb{G}_2^2$
    Return $e(\mathbf{c}_0, \mathbf{d}_0) \overset{?}{=} e(c_1^{y_1} \cdots c_n^{y_n}, d_1) \neq 1$

$\mathsf{FE.Enc}(\mathbf{x}, \mathsf{mpk})$:
    $\sigma, s \leftarrow^{\$} \mathbb{F}_q \setminus \{0\}$
    $c_i \leftarrow \left[\sigma x_i + s\mathbf{a}^\top \mathbf{w}_i\right]_1$
    Return $c \leftarrow \left([s\mathbf{a}]_1, c_1, \ldots, c_n\right)$.
```

Fig. 3. Our simplified version of [Wee17] FE scheme for orthogonality

(m_1, n_1), can only ask secret keys for keywords γ such that $\gamma \neq m_0 + \delta n_0$ and $\gamma \neq m_1 + \delta n_1$ for all $\delta \in [\kappa]$. This restriction (often referred to as weak attribute-hiding) is sufficient in our application as we want to hide the winner's index m mod n only from those users that haven't won i.e. from those holding keys for $\gamma \neq m \mod n$.

Concretely, we instantiate the construction in Fig.2, with a modified variant of the pairing-based FE for orthogonality proposed by Wee in [Wee17]. Our modified scheme is detailed in Figure 3. In the full version we prove the following theorem.

Proposition 1. *The scheme in Fig. 3 is selective secure under the* SXDH *assumption*

2.6 Non Interactive Zero-Knowledge

A non-interactive zero-knowledge (NIZK) proof system for a relation $\mathcal{R}$ is a tuple of PPT algorithms $(\mathsf{NIZK.G}, \mathsf{NIZK.P}, \mathsf{NIZK.V})$ where: $\mathsf{NIZK.G}$ generates a common reference string crs; $\mathsf{NIZK.P}(\mathsf{crs}, x, w)$, given $(x, w) \in \mathcal{R}$, outputs a proof π; $\mathsf{NIZK.V}(\mathsf{crs}, x, \pi)$, given statement x and proof π outputs 0 (reject) or 1 (accept). We say that a NIZK for $\mathcal{R}$ is *correct* if for every $\mathsf{crs} \leftarrow^{\$} \mathsf{NIZK.G}(1^\lambda)$ and all $(x, w) \in \mathcal{R}$, $\mathsf{NIZK.V}(\mathsf{crs}, x, \mathsf{NIZK.P}(\mathsf{crs}, x, w)) = 1$ holds with probability 1. In our protocols we require the NIZKs to satisfy the notions of weak simulation extractability [Sah99] and zero-knowledge [FLS90].

About the first property, it only guarantees the extractability of proofs produced by the adversary that are not equal to proofs previously observed. For this reason we make them "unique" by adding implicitly a session ID to the statement. Concretely this means that in the Fiat Shamir transform, the hash function evaluations need to be salted with a unique session ID. Note that we won't detail how to handle these *sid* (and neither we do this for ideal functionalities invocations).

We now define three relations about group elements. The first one checks whether two vectors $\mathbf{g}, \mathbf{h} \in \mathbb{G}_1^n$ are proportional, i.e., there exists $x \in \mathbb{F}_q$ s.t. $\mathbf{g}^x = \mathbf{h}$. The second one generalizes the previous to linear maps. The third one asks for solutions to the linear system $A\mathbf{x} = \mathbf{b}$ where A, $\mathbf{b}$ are given in the exponent and the last component x_n lies in a prescribed range. Formally

$$\begin{aligned}
\mathcal{R}_{\mathsf{DDH}} &= \{((\mathbf{g}, \mathbf{h}), x) \ : \ \mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \ \mathbf{g}^x = \mathbf{h}\} \\
\mathcal{R}_{\mathsf{Lin}} &= \left\{(([A]_1, [B]_1), X) \ : \ A \in \mathbb{F}_q^{k,m}, \ B \in \mathbb{F}_q^{k,n}, \ X \in \mathbb{F}_q^{m,n}, \ AX = B\right\} \\
\mathcal{R}_{\mathsf{LR}} &= \{(([A]_1, [\mathbf{b}]_1, R), \mathbf{x}) \ : \ A \in \mathbb{F}_q^{m,n}, \ \mathbf{b} \in \mathbb{F}_q^{m}, \ \mathbf{x} \in \mathbb{F}_q^{n}, \ A\mathbf{x} = \mathbf{b}, \ x_n \in [R]\}
\end{aligned}$$

We also use $\mathcal{R}_{\mathsf{Enc}}$ and $\mathcal{R}_{\mathsf{Dec}}$ which relates to a given functional encryption scheme. The first one, given a ciphertext, requires knowledge of the message and randomness used to generate it. The second one instead, given a tuple (mpk, c, f, x) asks for a *correct secret key* sk_f that decrypts c to x. Below we also introduce a language $\mathcal{L}_{\mathsf{key}}$ to formally capture the notion of correct secret key.

$$\begin{aligned}
\mathcal{L}_{\mathsf{key}} &= \{(\mathsf{mpk}, f, \mathsf{sk}) : \forall m, r; \ c = \mathsf{FE.Enc}(m, \mathsf{mpk}; r) \ \Rightarrow \ \mathsf{FE.Dec}(c, f, \mathsf{mpk}, \mathsf{sk}) = f(m)\} \\
\mathcal{R}_{\mathsf{Dec}} &= \{((\mathsf{mpk}, c, f, x), \mathsf{sk}) : (\mathsf{mpk}, f, \mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}, \ \mathsf{FE.Dec}(c, f, \mathsf{mpk}, \mathsf{sk}) = x\} \\
\mathcal{R}_{\mathsf{Enc}} &= \{((c, \mathsf{mpk}), (m, r)) : c = \mathsf{FE.Enc}(m, \mathsf{mpk}; r)\}.
\end{aligned}$$

Notice that, by abusing notation, standard asymmetric encryption, being a special case of FE, is also captured by this definition.

To construct our protocols, we assume the existence of a NIZK argument for each of these relations. We note that all of them can be proved through a sigma protocol, and that Fiat-Shamir based NIZKs from sigma protocols are weakly-simulation-extractable [FKMV12] based on a special property called quasi-unique responses. For the relations $\mathcal{R}_{\mathsf{DDH}}$ and $\mathcal{R}_{\mathsf{Lin}}$, we can use generalised Schnorr protocols provided in [Mau15]. For $\mathcal{R}_{\mathsf{LR}}$ we propose a variant of the folklore solution based on binary decomposition[11], in the full version. Still in the full version a sigma protocol for $\mathcal{R}_{\mathsf{Dec}}$ appears in the appendix.

2.7 UC model and Ideal Functionalities

The celebrated UC model, introduced in the seminal work of Ran Canetti [Can01], is a framework that allows to prove security properties of a protocol that are preserved under composition. This is done by comparing the protocol to an ideal functionality $\mathcal{F}$ defined to capture the intended properties. A protocol securely realises $\mathcal{F}$ if it is indistinguishable from $\mathcal{F} \circ \mathcal{S}$ for a given PPT simulator $\mathcal{S}$. The distinguisher $\mathcal{Z}$, also called the *environment*, is granted the power to choose all parties' input, learn their output and corrupt any number of parties learning their internal state and influencing their behavior. The challenge for $\mathcal{S}$ is

[11] In this case the most efficient choice to date may be an adaptation of Bulletproofs [BBB$^+$18]; however, to the best of our knowledge, this is only known to be simulation-extractable in the AGM [GOP$^+$21]. We leave the exploration of this optimization for future work.

therefore to reproduce all the messages sent by uncorrupted parties in a consistent way with their input/output, even though $\mathcal{S}$ cannot access it. To make this possible in non trivial cases, functionalities are often designed to leak some information to $\mathcal{S}$ and allow the simulator to influence the result in some way.

In Figure 4 we define two functionalities required in our construction: $\mathcal{F}_{\mathsf{zk}}$ and $\mathcal{F}_{\mathsf{CT}}^{\mathcal{D}}$ which respectively models a zero-knowledge proof of knowledge and a random beacon. The first one was introduced in [CF01], with the minor difference that in our case all the parties receive the output messages, deviation justified under the assumption of an authenticated broadcast channel. $\mathcal{F}_{\mathsf{CT}}$ instead was introduced in [CD20] and realised assuming honest majority under standard assumptions. We remark that our use of the random oracle for the NIZK proofs is justified assuming a global random oracle in the GUC model [CJS14, CDG⁺18]. Finally, about our communication model, we assume an authenticated broadcast channel with known bounded delay [KMTZ13], which implies that messages sent in broadcast are eventually delivered with potentially different order. Although this introduce some degree of synchronicity, this is in line with previous work [BEHG20].

The ZK Functionality $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}}$:
Upon receiving $(\mathsf{prove}, sid, x, w)$ from P_i, with sid being used by P_i for the first time: if $(x, w) \in \mathcal{R}$, broadcast $(\mathsf{proof}, sid, i, x)$.

The Coin Tossing Functionality $\mathcal{F}_{\mathsf{CT}}^{\mathcal{D}}$:
Parametrized by a distribution $\mathcal{D}$. Upon receiving (toss, sid) from all the honest parties, sample $x \xleftarrow{\$} \mathcal{D}$ and broadcast $(\mathsf{tossed}, sid, x)$

Fig. 4. Description of the functionalities $\mathcal{F}_{\mathsf{zk}}$ and $\mathcal{F}_{\mathsf{CT}}$

3 Universally Composable SSLE

The notion of single secret leader election was introduced in [BEHG20] as a tuple of protocols (SSLE.Setup, SSLE.Reg, SSLE.Elect, SSLE.Claim, SSLE.Vrf) aimed at electing a unique leader among a set of participants who can stay hidden until she decide to reveal herself. Security of this primitive was captured through three game-based properties, namely *uniqueness*, *fairness* and *unpredictability*. However, the underlying security experiments fail to capture scenarios where multiple executions of the given procedures may occur concurrently. Moreover, as in most game-based notions, security is not guaranteed to hold when the primitive is used in a more complex protocol.

For this reason, we propose a definition of SSLE in the universal composability model. To this end, we define a functionality $\mathcal{F}_{\mathsf{SSLE}}$ that performs elections and reveals the winners in an ideal way. A UC-secure SSLE scheme is then any protocol that securely realizes $\mathcal{F}_{\mathsf{SSLE}}$.

At a high-level, $\mathcal{F}_{\mathsf{SSLE}}$ consists of the following commands. By using $(\mathsf{register})$ a user can register to an election. When all the honest users call (elect, eid), a new election with identifier eid is performed, that is, the ideal functionality samples a winner index j uniformly at random from the set of registered users. By using the (elect, eid) command, every honest user is informed by the ideal functionality on whether she is the winner of the election eid. Using (reveal, eid), an honest winning user instructs the ideal functionality to announce the election's outcome to everyone. Finally, the $(\mathsf{fake\_rejected}, eid, j)$ command is reserved to the adversary and makes $\mathcal{F}_{\mathsf{SSLE}}$ announce to everyone that P_j is *not* the winner. This models a scenario in which an adversary who won an election deviates from the protocol to claim victory in spite of being the winning leader. The formal definition $\mathcal{F}_{\mathsf{SSLE}}$ is detailed in Figure 5.

The SSLE functionality $\mathcal{F}_{\mathsf{SSLE}}$:
Initialise $E, R \leftarrow \varnothing$, $n \leftarrow 0$ and let $M \subseteq [N]$ be the set of corrupted parties. Upon receiving:

- $(\mathsf{register})$ from P_i: add $R \leftarrow R \cup \{(i, n)\}$, broadcast $(\mathsf{registered}, i)$ and set $n \leftarrow n + 1$.
- (elect, eid) from all honest parties: if $R \neq \varnothing$ and eid was not requested before, sample $(j, \gamma) \xleftarrow{\$} R$ and send $(\mathsf{outcome}, eid, 1)$ to P_j and $(\mathsf{outcome}, eid, 0)$ to P_i for $(i, \cdot) \in R$, $i \neq j$. Store $E \leftarrow E \cup \{(eid, j)\}$.
- (reveal, eid) from P_i: if $(eid, i) \in E$ broadcast $(\mathsf{result}, eid, i)$. Otherwise broadcast $(\mathsf{rejected}, eid, i)$.
- $(\mathsf{fake\_rejected}, eid, j)$ from $\mathcal{Z}$: if P_j is corrupted broadcast $(\mathsf{rejected}, eid, j)$.

Fig. 5. SSLE functionality executed among $P_1, \ldots, P_N$ and environment $\mathcal{Z}$

In order to capture constructions that are secure against adversaries capable of corrupting only a fraction of the participants, we distinguish two thresholds parameters: $t \in [N]$, which bounds corruptions among *all* users $P_1, \ldots, P_N$ (even those who are not registered), and $\vartheta : \mathbb{N} \to \mathbb{N}$, which upper bounds corruptions among the set of *currently registered* users depending on their number. Even though this notation is non-standard, it allows us to formalise standard assumptions such as the existence of an honest majority among *currently active* users, which is employed in several blockchain protocols. More formally, we give the following definitions.

Definition 5. *Let $t \in [N]$ and $\vartheta : \mathbb{N} \to \mathbb{N}$. A protocol Π is said to statically (t, ϑ)-threshold realise $\mathcal{F}_{\mathsf{SSLE}}$ if there exists a simulator $\mathcal{S}$ such that Π is indistinguishable from $\mathcal{F}_{\mathsf{SSLE}} \circ \mathcal{S}$ for all* PPT *environments $\mathcal{Z}$ that statically corrupt a set M of parties with $|M| < t$ and such that at each step, calling R the set of registered users, $|R \cap M| < \vartheta(|R|)$.*

Definition 6. *A* (t, ϑ)*-**threshold statically secure UC-SSLE** is a protocol* Π *that* (t, ϑ)*-securely realise* $\mathcal{F}_{\mathsf{SSLE}}$*. If* $t = N$ *and* $\vartheta = 1_{\mathbb{N}}$ *then* Π *is called a **statically secure UC-SSLE**.*

To further motivate our UC-secure notion of SSLE we compare it to the game-based one. First, with the following proposition, we show that the UC notion implies the game-based one. For the sake of generality we informally say that a property is (t, ϑ)*-threshold satisfied* if it holds against an adversary that corrupts at most t users and up to $\vartheta(|R|)$ of them belong to the set R of those currently registered at each step. For a more formal treatment see the full version of this paper, where also a proof of the following appears.

Proposition 2. *If* Π *is a* (t, ϑ)*-threshold statically secure UC-SSLE protocol, then its derived SSLE scheme described in Figure 6 satisfies* (t, ϑ)*-threshold uniqueness,* (t, ϑ)*-threshold fairness and* (t, ϑ)*-threshold unpredictability.*

```
$\mathsf{SSLE.Setup}$:
    $P_i$ sets $(\mathsf{pp}, \mathsf{sp}_i, eid) \leftarrow (\bot, \bot, 0)$

$\mathsf{SSLE.Reg}_{\mathsf{pp}}(i)$:
    $P_i$ sends $(\mathsf{register}, i)$ to $\Pi$
    Others wait $(\mathsf{registered}, i) \leftarrow \Pi$

$\mathsf{SSLE.Elect}_{\mathsf{pp}}$:
    All players send $(\mathsf{elect}, eid)$ to $\Pi$ and update $eid \leftarrow eid + 1$
    When $P_i$ receives $(\mathsf{outcome}, eid, \cdot) \leftarrow \Pi$: return $eid$

$\mathsf{SSLE.Claim}_{\mathsf{pp}}(c, \mathsf{sp}_i, i)$:
    Send $(\mathsf{reveal}, c)$ to $\Pi$.
    Return $\pi \leftarrow \bot$

$\mathsf{SSLE.Vrf}_{\mathsf{pp}}(c, \pi, i)$:
    When $(\mathsf{result}, c, i) \leftarrow \Pi$ return 1
    When $(\mathsf{rejected}, c, i) \leftarrow \Pi$ return 0
```

Fig. 6. The derived SSLE scheme from a UC-SSLE protocol Π

Second, we argue that our UC notion is strictly stronger than the game-based one. For this, we simply observe that taking one of the protocols from [BEHG20] (e.g., the one based on TFHE or the one based on Shuffling) they cannot be UC-secure if the zero-knowledge proofs they employ are not UC-secure.[12] In [BEHG20], these protocols are proven secure without making any UC assumption on these zero-knowledge proofs; so they constitute a counterexample of protocols that are secure in the game-based sense but would not be secure according to our UC notion.

[12] Here, as the candidate protocol we are assuming the one where each sub protocol is used to implement the corresponding command, i.e., SSLE.Reg for register, SSLE.Elect for elect, etc.

3.1 A parametrised definition

Definition 6 provides a higher level of security with respect to the game-based definition in [BEHG20], but at the same time requires more structure from the underlying protocol and therefore may imply higher costs. In order to leverage security and efficiency we present here a "tunable" functionality $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$ which allows the adversary to control, with probability smaller than $2^{-\kappa}$, a given election and which may not elect any user with probability smaller than $2^{-\eta}$.

The Parametrised SSLE functionality $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$:
Initialise $E, R \leftarrow \varnothing, n \leftarrow 0$ and let $M \subseteq [N]$ be the set of corrupted parties. Upon receiving:

- $(\mathsf{register})$ from P_i: add $R \leftarrow R \cup \{(i,n)\}$, broadcast $(\mathsf{registered}, i)$ and set $n \leftarrow n+1$.
- (elect, eid) from honest parties: if eid was not requested before leak to $\mathcal{Z}$ $(\mathsf{electing}, eid)$. Upon receiving $(\mathsf{prob}, eid, p_1, p_2)$ with $p_1 \leq 2^{-\kappa}, p_2 \leq 2^{-\eta}$:

  With probability p_1 leak $(\mathsf{corrupted}, eid)$ and wait for the adversary to reply with (infl, eid, j). Else, with probability p_2 set $j \leftarrow \perp$. If the previous actions are not performed, sample $(j, \cdot\,) \xleftarrow{\$} R$.

  Send $(\mathsf{outcome}, eid, 1)$ to P_j and $(\mathsf{outcome}, eid, 0)$ to P_i for $(i, \cdot\,) \in R$, $i \neq j$. Add $E \leftarrow E \cup \{(eid, j)\}$.
- (reveal, eid) from P_i: if $(eid, i) \in E$ broadcast $(\mathsf{result}, eid, i)$. Otherwise broadcast $(\mathsf{rejected}, eid, i)$.
- $(\mathsf{fake\_rejected}, eid, j)$ from $\mathcal{Z}$: if P_j is corrupted broadcast $(\mathsf{rejected}, eid, j)$.

Fig. 7. Parametrised SSLE executed among $P_1, \ldots, P_N$ and environment $\mathcal{Z}$

Setting $\kappa = \eta = \Theta(\lambda)$ we get back a functionality equivalent to $\mathcal{F}_{\mathsf{SSLE}}$. However for smaller κ, η, we can now capture schemes achieving weaker (but still meaningful!) fairness and unpredictability notions. These might be acceptable/sufficient in practical scenarios, especially if they lead to significant efficiency gains. In the full version we show that applying the construction in Figure 6 to a protocol realizing $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$ yields an SSLE scheme with $(2^{-\kappa} + 2^{-\eta})$-fairness and $\xi(\kappa)$-unpredictability with

$$\xi(\kappa) \;=\; \sup_{n \in \mathbb{N}} \left(\frac{n}{n - \vartheta(n)} \right) \cdot \frac{1}{2^{\kappa}} \cdot \frac{2^{\eta}}{2^{\eta} - 1}.$$

For fairness, the $2^{-\kappa} + 2^{-\eta}$ bound simply means that for $\kappa, \eta = \log N$ an adversary controlling T parties, wins the election with probability $(T+2)/N$. This is the same winning probability of an adversary that runs a perfectly fair election but corrupts two single extra players.

4 UC-secure SSLE from FE for Modular Keyword Search

We now present a generic construction of a UC-SSLE protocol based on modular keyword search FE. This, besides being of interest on its own, serves as a warm-up for our efficient construction of Section 5. More specifically, assuming for the sake of abstraction the existence of a protocol Π which securely distributes keys and, on request, produces ciphertexts encrypting random messages in a given set, we UC-realise $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$.

Our construction roughly works as follows: Initially the public key mpk is distributed among the N users. To perform the n-th registration for P_i, parties run Π to give sk_n to P_i. When an election is requested, users generate with Π a challenge ciphertext c that encrypts a message m, n, with $m \in [\kappa n]$ such that $m \bmod n \sim U([n])$, and check whether they won or lost by decrypting. Whoever can decrypt c to 1 is the leader and can claim victory by broadcasting a NIZK argument of this.

Unfortunately, even if this solution can already be proven secure in the game-based definition, it is not UC-secure yet. The reason is technical: if at a given round a ciphertext c encrypting (m, n) with $m = \gamma + \delta n$ is returned, γ being associated to an honest user[13], the adversary could re-register malicious users until he gets sk_m and then test that $\mathsf{MKS.Dec}(c, m, \mathsf{mpk}, \mathsf{sk}_m) = 1$. This makes the protocol hard to simulate as the ciphertext produced needs to always contain the winner's index – which the simulator may not know in advance[14].

To prevent this issue we introduce a set S of forbidden keys: each time a user wins with key sk_γ, the indices $\gamma + \delta n$ for $\delta \in [\kappa]$ are added to S and, each time a new user joins, n is set to be the next integer not lying in S. However this introduce a probability $|[n] \cap S| \cdot n^{-1}$ to produce a ciphertext no one can decrypt, meaning that nobody is elected. A way to keep it smaller than $2^{-\eta}$ is to perform a new setup every time the above probability exceeds this bound.

To proceed we formally define a functionality $\mathcal{F}_{\mathsf{SnC}}$, Fig. 5, which shapes the behaviour and security of Π, and a protocol $\{P_{\mathsf{MKS-SSLE}}^{(i)} : i \in [N]\}$ in the $\mathcal{F}_{\mathsf{SnC}}$-hybrid model realising $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$. A proof of security appears in the full version.

Theorem 1 *The protocol $\{P_{\mathsf{MKS-SSLE}}^{(i)} : i \in [N]\}$ in Figure 9 securely realises $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$ in the $\mathcal{F}_{\mathsf{SnC}}$-hybrid model for the class of* PPT *environments $\mathcal{Z}$ that statically corrupts up to N players for any positive κ, η.*

5 An Efficient UC-secure SSLE from SXDH

In this section we present our main contribution, an SSLE protocol that works over bilinear groups which we prove UC-secure under the SXDH assumption.

[13] i.e. such that an honest user P_i posses the key sk_γ

[14] When $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$ elects an honest user, the simulator learn its identity only after this party is instructed by the environment to claim victory through a reveal command.

The Setup and Challenge Functionality $\mathcal{F}_{\mathsf{SnC}}$:

Generate $\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.G_{Dec}}(1^\lambda)$, $\mathsf{mpk}, \mathsf{msk} \xleftarrow{\$} \mathsf{KS.Setup}(1^\lambda)$. Set $S \leftarrow \varnothing$ and $n \leftarrow 0$. Upon receiving:

- (setup) from P_i: send $(\mathsf{input}, \mathsf{crs}, \mathsf{mpk})$ to P_i.
- $(\mathsf{ch\_request}, eid)$ from all honest parties: If eid was not requested before, sample $m \xleftarrow{\$} [\kappa n]$ and compute $c \xleftarrow{\$} \mathsf{KS.Enc}((m, n), \mathsf{mpk})$. Broadcast $(\mathsf{challenge}, eid, c)$. If $m \notin S$, divide $m = \gamma + \delta n$ with $\gamma \in [n]$ and add $S \leftarrow S \cup \{\gamma + \delta' n \,:\, \delta' \in [\kappa]\}$.
- (keygen) from P_i: When honest users agree, $n \leftarrow n + 1$ until $n \notin S$, set $\mathsf{sk}_n \leftarrow \mathsf{MKS.KeyGen}(n, \mathsf{msk})$ send $(\mathsf{key}, \mathsf{sk}_n)$ to P_i and broadcast $(\mathsf{key\_request}, i, n)$.

Fig. 8. Setup and Challenge functionality executed among $P_1, \ldots, P_N$

5.1 Intuition

At a high level we instantiate the generic protocol provided in the previous section with the modular KS scheme obtained applying the transformation in Fig. 2 to our OFE in Fig. 3. The main challenge here is to efficiently generate ciphertexts in a distributed way. To address this, the basic idea is to select a random committee $Q \subseteq [N]$, have each member P_j secretly sample a value $m_j \in [n]$, where n is the number of users currently registered, and jointly generate an encryption of $m = \sum_{j \in Q} m_j$.

A downside is that now $m \in [|Q| \cdot n]$. For this reason we set $\kappa \geq |Q|$, where κ parametrises the set of functions $\mathsf{F}^{\kappa}_{\mathsf{mks}}$ supported by our modular KS scheme. In this way, as in the generic construction, decryption allows the holder of a secret key sk_γ to learn only whether $m = \gamma \bmod n$ or not. Also, if at least one m_j is uniform over $[n]$, so is $m \bmod n$, implying that the election is fair. Finally, since $|Q| = \kappa$ is a small parameter, the decryption procedure in our scheme (Fig. 2) remains efficient.

The next step is to show in more detail how the committee can accomplish its task. The ciphertext we want to produce is the encryption of $(m, -1, -n)$ under our OFE scheme in Fig. 3 and has the following form

$$\mathbf{c}_0 = [s\mathbf{a}]_1\,,\ c_1 = \left[\sigma m + s\mathbf{a}^\top \mathbf{w}_1\right]_1,\ c_2 = \left[-\sigma + s\mathbf{a}^\top \mathbf{w}_2\right]_1,\ c_3 = \left[-n\sigma + s\mathbf{a}^\top \mathbf{w}_3\right]_1$$

with $s, \sigma \sim U(\mathbb{F}_q)$ and $[\mathbf{a}]_1\,,\ \left[\mathbf{a}^\top \mathbf{w}_\ell\right]_1$ being the public key. While $\mathbf{c}_0, c_2, c_3$ would be easy to generate in a distributed way, as they linearly depend on s, σ, in c_1 we need to compute a product $\sigma \cdot m$. Standard MPC techniques could solve this issue within a few rounds, however we opt for a solution that requires each user to only speak once.

First, we sample two group elements G, H through the random beacon and interpret them as the ElGamal encryption, with respect to a previously generated public key g, h, of $[\sigma]_1$. Next each player P_i for $i \in Q$ samples $m_i \in [n]$, $s_i \in [n]$, and, using the linearity of ElGamal, computes and randomise an encryption of

$$c_{1,i} = \left[\sigma m_i + s_i \mathbf{a}^\top \mathbf{w}_1\right]_1, \quad c_{2,i} = \left[-\sigma + s_i \mathbf{a}^\top \mathbf{w}_2\right]_1, \quad c_{3,i} = \left[-n\sigma + s_i \mathbf{a}^\top \mathbf{w}_3\right]_1$$

Party $P^{(i)}_{\mathsf{MSK-SSLE}}$ realising $\mathcal{F}^{\kappa,\eta}_{\mathsf{SSLE}}$:

Set $C, R, \mathcal{K} \leftarrow \varnothing$, send setup to $\mathcal{F}_{\mathsf{SnC}}$ and wait for $(\mathsf{input}, \mathsf{crs}, \mathsf{mpk})$. Upon receiving:

- $\mathsf{register}$: send keygen to $\mathcal{F}_{\mathsf{SnC}}$ and wait for its reply $(\mathsf{key}, \mathsf{sk}_\gamma)$. Store $\mathcal{K} \leftarrow \mathcal{K} \cup \{\mathsf{sk}_\gamma\}$.
- A request to generate a new secret key from $\mathcal{F}_{\mathsf{SnC}}$: accepts if there is no election currently in progress.
- $(\mathsf{key\_request}, j, n)$ from $\mathcal{F}_{\mathsf{SnC}}$: return $(\mathsf{registered}, j)$ and set $R \leftarrow R \cup \{(j, n)\}$.
- (elect, eid): send $(\mathsf{ch\_request})$ to $\mathcal{F}_{\mathsf{SnC}}$. When it replies with $(\mathsf{challenge}, eid, c)$, if there exists $\mathsf{sk}_\gamma \in \mathcal{K}$ such that $1 \leftarrow \mathsf{MKS.Dec}(c, \gamma, \mathsf{mpk}, \mathsf{sk}_\gamma)$, return $(\mathsf{outcome}, eid, 1)$. Otherwise return $(\mathsf{outcome}, eid, 0)$. Add $C \leftarrow C \cup \{(c, eid)\}$.
- (reveal, eid): if $(eid, c) \in C$ and $1 \leftarrow \mathsf{KS.Dec}(c, i, \mathsf{mpk}, \mathsf{sk}_\gamma)$ for some $\mathsf{sk}_\gamma \in \mathcal{K}$, prove $\pi \xleftarrow{\$} \mathsf{NIZK.P}_{\mathsf{Dec}}(\mathsf{mpk}, c, \gamma, \mathsf{sk}_\gamma)$ and broadcast $(\mathsf{claim}, eid, \pi, \gamma)$. Otherwise broadcast $(\mathsf{claim}, eid, \perp)$.
- $(\mathsf{claim}, eid, \pi, \gamma)$ from P_j: if $(j, \gamma) \in R$ and $1 \leftarrow \mathsf{NIZK.V}_{\mathsf{Dec}}(\mathsf{crs}, \mathsf{mpk}, c, \gamma, \pi)$ return $(\mathsf{result}, eid, j)$, otherwise $(\mathsf{rejected}, eid, j)$

Fig. 9. Reduction of $\mathcal{F}^{\kappa,\eta}_{\mathsf{SSLE}}$ to the Setup and Challenge functionality $\mathcal{F}_{\mathsf{SnC}}$

Finally he publish these encrypted values together with $\mathbf{c}_{0,i} = [s_i\mathbf{a}]_1$ in plain and a NIZK. At this point everyone can locally set $\mathbf{c}_0$ as the product of the $\mathbf{c}_{0,i}$'s and compute ElGamal encryptions of c_1, c_2, c_3 that are respectively the products of $c_{1,i}$, $c_{2,i}$ and $c_{3,i}$. The last step would then be to decrypt these three remaining components. To this aim we assume that the secret key x of the ElGamal public key $h = g^x$ was previously t-shared among all users, which allows us to perform a threshold decryption.

To complete the protocol we have to show how to distribute the setup and key generation of our FE scheme. For ease of exposition, we first present a protocol assuming an ideal setup functionality in Section 5.2, and then in Section 5.3 we show how this functionality can be UC-realized. In conclusion we point out that, as in the general construction in Section 4, we have to maintain a set S of keys that cannot be generated in order to keep the protocol simulatable, resulting occasionally in elections without leaders.

5.2 SSLE protocol with Ideal Setup Functionality

In Figure 11 we show a protocol that securely realizes the $\mathcal{F}^{\kappa,\eta}_{\mathsf{SSLE}}$ ideal functionality. To this end we use the following building blocks:

- The FE scheme for orthogonality in Fig. 3, denoted FE which we use to instantiate a modular KS scheme.
- NIZKs for $\mathcal{R}_{\mathsf{DDH}}, \mathcal{R}_{\mathsf{LR}}$ and $\mathcal{R}_{\mathsf{Dec}}$. For readability, we suppress the crs from the inputs of the prover and verifier algorithm.

– A functionality $\mathcal{F}_{\mathsf{SK}}$ that distributes public and private keys of our OFE scheme, and t-share a threshold ElGamal secret key – sending privately the share $f(j)$ to P_j and publicly $k_j = g^{f(j)}$.
– A random beacon $\mathcal{F}_{\mathsf{CT}}^{\mathsf{ch}}$ returning G, H, Q with $G, H \sim U(\mathbb{G}_1^2)$ and $Q \subseteq [N]$, $|Q| = \ell$ such that the probability that Q is contained in the set of corrupted parties is smaller than $2^{-\kappa}$. Note that $t < N/2$ implies $\ell \leq \kappa$.

Each user maintains (or recovers from the public state) four sets $C, R, S, \mathcal{K}$ respectively containing previous challenges, currently registered users, forbidden keys and owned secret keys.

Elections begin by invoking $\mathcal{F}_{\mathsf{CT}}^{\mathsf{ch}}$ which returns (G, H, Q). In steps 6-8 users in Q interpret $(G, H) = (g^\theta, h^\theta \cdot [\sigma]_1)$ as an ElGamal encryption with $\sigma \sim U(\mathbb{F}_q)$, sample $m_i \in [n]$, $s_i \in \mathbb{F}_q$ and produce encrypted shares of the challenge components. Then they sample r_i and ρ_i to re-randomize these ciphertexts. Interestingly we observe that using the same randomness for the last two components does not affect security.

Next, in steps 11-15 we let $Q_0 \subseteq Q$ be the set of users who replied with a correct NIZK. Observe that, calling s, r, ρ, m the sum of the respective shares s_i, r_i, ρ_i and m_i over Q_0, then $G_1 = g^{r+\theta m}$ and $G_2 = g^{\rho+\theta}$. In order to decrypt each user produces $K_{1,i}, K_{2,i}$ that will open, through a Shamir reconstruction in the exponent, to $h^{r+\theta m}$ and $h^{\rho+\theta}$.

In steps 16-20, users locally multiply the elements sent by the committee and reconstruct, interpolating at the exponent, $K_1 = h^{r+\theta m}$ and $K_2 = h^{\rho+\theta}$. Since

$$\prod_{\mu \in Q_0} c_{1,\mu} = h^{r+\theta m} \left[m\sigma + s\mathbf{a}^\top \mathbf{w}_1\right]_1 \qquad H^{-1} \prod_{\mu \in Q_0} c_{2,\mu} = h^{-\rho-\theta} \left[-\sigma + s\mathbf{a}^\top \mathbf{w}_2\right]_1$$

$$H^{-n} \prod_{\mu \in Q_0} c_{3,\mu} = h^{-n(\rho+\theta)} \left[-n\sigma + s\mathbf{a}^\top \mathbf{w}_3\right]_1$$

applying K_1, K_2 they finally obtain all the components of the challenge c. At the end of an election (lines 20-22) each user verify whether or not he won by attempting to decrypt the produced challenge with the keys he stored in $\mathcal{K}$.

When a user wins and is instructed through a reveal command to claim victory, he sends both the elements K_1, K_2 previously computed and a proof of knowledge of a secret key $\mathsf{sk}_{\gamma,j}$ which decrypts the challenge to 1. The first part is required as we don't want to store on chain the threshold decryption. This may sound insecure at first, as another user could come up with different K_1', K_2' that let him win. Interestingly, in the proof of security we show that being able to do so implies breaking the selective security of our OFE.

Theorem 2 *The protocol in Fig. 11 (t, ϑ)-threshold securely realizes $\mathcal{F}_{\mathsf{SSLE}}^{\kappa,\eta}$ in the $(\mathcal{F}_{\mathsf{CT}}, \mathcal{F}_{\mathsf{SK}})$-hybrid model under the* SXDH *assumption, for $t = \lfloor N/2 \rfloor$ and $\vartheta(n) = \lfloor n/2 \rfloor$*

The Setup and Key Generation Functionality $\mathcal{F}_{\mathsf{SK}}$
$n \leftarrow 0$, $S \leftarrow \varnothing$, $g \xleftarrow{\$} \mathbb{G}_1$, $f \xleftarrow{\$} \mathbb{F}_q[x]_{<t}$, $\mathsf{mpk}, \mathsf{msk} \xleftarrow{\$} \mathsf{FE.Setup}(1^\lambda, 3)$. Fix $h \leftarrow g^{f(-1)}$, $k_j \leftarrow g^{f(j)}$, $\mathsf{pp} \leftarrow (\mathsf{mpk}, g, h, (k_j)_{j=0}^{N-1})$, leak $(\mathsf{setup\_leak}, \mathsf{pp}, f(j))_{j\in M}$ and wait for $(\mathsf{setup\_infl}, \mathbf{w}^*_\alpha, f^*)_{\alpha=1}^3$ from the adversary $\mathcal{Z}$. Calling $\mathsf{mpk} = (\mathbf{z}_0, z_\alpha)_{\alpha=1}^3$, $\mathsf{msk} = (\mathbf{w}_\alpha)_{\alpha=1}^3$, set

$$\mathsf{mpk} \leftarrow (\mathbf{z}_0, z_\alpha \cdot \mathbf{z}_0^{\mathbf{w}^*_\alpha})_{\alpha=1}^3, \qquad \mathsf{msk} \leftarrow (\mathbf{w}_\alpha + \mathbf{w}^*_\alpha)_{\alpha=1}^3, \qquad f \leftarrow f + f^*$$

and update pp. Upon receiving:

- (setup) from P_j: Send $(\mathsf{input}, \mathsf{pp}, f(j))$ to P_j.
- $(\mathsf{update}, \gamma, n)$ from honest players: $S \leftarrow S \cup \{\gamma + \delta n \,:\, \delta \in [\kappa]\}$.
- (keygen) from P_j: Broadcast $(\mathsf{key\_request})$. while $n \in S$, increase n by 1. Set $\mathsf{sk}_{n,\delta} \leftarrow \mathsf{FE.KeyGen}(\mathbf{y}_{n,\delta}, \mathsf{msk})$ and send $(\mathsf{key}, (\mathsf{sk}_{n,\delta})_{\delta=0}^{\kappa-1})$ to P_j.
- $(\mathsf{infl}, (\mathbf{w}^*_i)_{i=1}^3)$ from the adversary $\mathcal{Z}$: For each key $\mathsf{sk} = (\mathbf{d}, d)$ sent to P_j associated to a vector $\mathbf{y}$, compute $\mathsf{sk}' = (\mathbf{d} \cdot [y_1\mathbf{w}^*_1 + y_2\mathbf{w}^*_2 + y_3\mathbf{w}^*_3]_d, d)$ and send $(\mathsf{key\_update}, \mathsf{sk}')$ to P_j. Update msk setting $\mathbf{w}_\alpha \leftarrow \mathbf{w}_\alpha + \mathbf{w}^*_\alpha$.

Fig. 10. Functionality $\mathcal{F}_{\mathsf{SK}}$ among users $P_1, \ldots, P_N$ and environment $\mathcal{Z}$ which in Protocol 11 performs the setup and distributes keys on request

5.3 Realising the Setup

Here we describe how to realize the functionality $\mathcal{F}_{\mathsf{SK}}$ deployed in Protocol 11.

First of all, in order to emulate private communication channels, not available in our model but necessary to distribute secret parameters, we use an IND-CPA encryption scheme ($\mathsf{AE.Setup}, \mathsf{AE.Enc}, \mathsf{AE.Dec}$). Second, as our NIZKs are randomised sigma protocols compiled with Fiat-Shamir, they only need access to a random oracle and in particular there is no need to instantiate a crs. Next, we need to distribute the secret key of the Threshold ElGamal scheme. This is addressed by deploying standard techniques from verifiable secret sharing.

Finally we have to generate the public and secret keys of the FE scheme in Figure 3. To this aim, recall that

$$\mathsf{mpk} = [\mathbf{a}]_1, ([\mathbf{a}^\top \mathbf{w}_\alpha]_1)_{\alpha=1}^3, \qquad \mathsf{sk}_{\mathbf{y}_{\gamma,\delta}} = [r(\mathbf{w}_1 + \gamma\mathbf{w}_2 + \delta\mathbf{w}_3)]_2, [r]_2.$$

Fixing $[\mathbf{a}]_1$ and $[r]_2$, which can be generated through a random beacon, the remaining components of these keys depends linearly on $\mathbf{w}_\alpha$. Therefore we can again select a random committee and let each member P_i sample $\mathbf{w}_{\alpha,i} \xleftarrow{\$} \mathbb{F}_q^2$. At a high level to produce either mpk or a secret key, users provide shares of it, which are then locally multiplied. When reconstructing a secret key moreover the receiver checks the shares and complain if they are malformed.

More in detail in our construction we will use

- NIZKs for $\mathcal{R}_{\mathsf{Enc}}, \mathcal{R}_{\mathsf{Dec}}$ and the ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Lin}}$.
- Two random beacons $\mathcal{F}_{\mathsf{CT}}^{\mathsf{stp}}$ and $\mathcal{F}_{\mathsf{CT}}^{\mathsf{sk}}$ returning respectively $(Q, \mathbf{z}_0, g)$ and $(d_\delta)_{\delta=0}^{\kappa-1}$ with $\mathbf{z}_0 \sim U(\mathbb{F}_q^2)$, $g \sim U(\mathbb{G}_1)$, $d_\delta \sim U(\mathbb{G}_2)$ and $Q \subseteq [N]$, $|Q| = \ell$

Party $P^{(i)}_{\mathsf{SSLE},\kappa}$ realising $\mathcal{F}^{\kappa,\eta}_{\mathsf{SSLE}}$:

```
Call $C, R, S, \mathcal{K} \leftarrow \varnothing$, $n \leftarrow 0$. Send $\mathsf{setup}$ to $\mathcal{F}_{\mathsf{SK}}$, wait for the reply $(\mathsf{input}, \mathsf{pp}, f(i))$
and parse $\mathsf{pp} = \mathsf{mpk}, g, h, k_0, \dots, k_{N-1}$, with $\mathsf{mpk} = [\mathbf{a}]_1, [\mathbf{a}^\top \mathbf{w}_1]_1, [\mathbf{a}^\top \mathbf{w}_2]_1,$
$[\mathbf{a}^\top \mathbf{w}_3]_1$ together with bilinear groups description. Upon receiving:

1 : $(\mathsf{register})$: Send $\mathsf{keygen}$ to $\mathcal{F}_{\mathsf{SK}}$ and wait for $(\mathsf{key}, \mathsf{sk}_n)$; Add $\mathcal{K} \leftarrow \mathcal{K} \cup \mathsf{sk}_n$
2 : $(\mathsf{key\_requested}, j)$ from $\mathcal{F}_{\mathsf{SK}}$: Wait for all elected leader to reveal themselves.
3 :   While $n \in S$, $n \leftarrow n+1$;   $R \leftarrow R \cup \{(j,n)\}$, $n \leftarrow n+1$;   Return $(\mathsf{registered}, j)$

4 : $(\mathsf{elect}, eid)$: Send $(\mathsf{toss}, eid)$ to $\mathcal{F}^{\mathsf{ch}}_{\mathsf{CT}}$
5 : $(\mathsf{tossed}, eid, G, H, Q)$ from $\mathcal{F}^{\mathsf{ch}}_{\mathsf{CT}}$, if $i \in Q$:
6 :   Sample $s_i, r_i, \rho_i \xleftarrow{\$} \mathbb{F}_q$ and $m_i \xleftarrow{\$} [n]$ and compute:
7 :     $G_{1,i} \leftarrow g^{r_i} G^{m_i}$,   $G_{2,i} \leftarrow g^{\rho_i}$,   $\mathbf{c}_{0,i} \leftarrow [s_i \mathbf{a}]_1$,   $c_{1,i} \leftarrow h^{r_i} H^{m_i} \cdot [s_i \mathbf{a}^\top \mathbf{w}_1]_1$
8 :     $c_{2,i} \leftarrow h^{-\rho_i} \cdot [s_i \mathbf{a}^\top \mathbf{w}_2]_1$,   $c_{3,i} \leftarrow h^{-n\rho_i} \cdot [s_i \mathbf{a}^\top \mathbf{w}_3]_1$
9 :     $\pi_{\mathsf{LR},i} \leftarrow \mathsf{NIZK.P}_{\mathsf{LR}}(\mathbf{S}, (\mathbf{c}_{0,i}, c_{1,i}, c_{2,i}, c_{3,i}, G_{1,i}, G_{2,i}), [n], (s_i, r_i, \rho_i, m_i))$
10 :   Broadcast $(\mathsf{msg}, eid, \mathbf{c}_{0,i}, c_{1,i}, c_{2,i}, c_{3,i}, G_{1,i}, G_{2,i}, \pi_{\mathsf{LR},i})$
11 : $(\mathsf{msg}, eid, \mathbf{c}_{0,\nu}, c_{1,\nu}, c_{2,\nu}, c_{3,\nu}, G_{1,\nu}, G_{2,\nu}, \pi_{\mathsf{LR},\nu})$ from $P_\nu$:
12 :   Let $Q_0 \subseteq Q$ be the set of $\nu$ such that $\pi_{\mathsf{LR},\nu}$ is accepted
13 :   $G_1 \leftarrow \prod_{\nu \in Q_0} G_{1,\nu}$,   $G_2 \leftarrow G \cdot \prod_{\nu \in Q_0} G_{2,\nu}$,   $K_{1,i} \leftarrow G_1^{f(i)}$,   $K_{2,i} \leftarrow G_2^{f(i)}$
14 :   $\pi_{\mathsf{DDH},i} \leftarrow \mathsf{NIZK.P}_{\mathsf{DDH}}((g, G_1, G_2), (k_i, K_{1,i}, K_{2,i}), f(i))$
15 :   Broadcast $(\mathsf{open}, eid, K_{1,i}, K_{2,i}, \pi_{\mathsf{DDH},i})$
16 : $(\mathsf{open}, eid, K_{1,\nu}, K_{2,\nu}, \pi_{\mathsf{DDH},\nu})$ with valid proof, from a set $Z \subseteq [N]$ of $t$ parties:
17 :   Reconstruct $K_j \leftarrow \prod_{\nu \in Z} K_{j,\nu}^{\lambda_\nu}$ with $\lambda_\nu$ the Lagrange coefficient for $Z$
18 :   $\mathbf{c}_0 \leftarrow \prod_{\mu \in Q_0} \mathbf{c}_{0,\mu}$,   $c_1 \leftarrow K_1^{-1} \cdot \prod_{\mu \in Q_0} c_{1,\mu}$,   $c_2 \leftarrow H^{-1} K_2 \cdot \prod_{\mu \in Q_0} c_{2,\mu}$
19 :   $c_3 \leftarrow H^{-n} K_2^n \cdot \prod_{\mu \in Q_0} c_{3,\mu}$,   $c \leftarrow (\mathbf{c}_0, c_1, c_2, c_3)$
20 :   If there exists $\mathsf{sk}_{\gamma,\delta} \in \mathcal{K}$ which decrypts $c$ to 1:
21 :     Return $(\mathsf{outcome}, eid, 1)$ and store $C \leftarrow C \cup \{(eid, K_1, K_2)\}$
22 :   Else return $(\mathsf{outcome}, eid, 0)$.

23 : $(\mathsf{reveal}, eid)$: If there exists $(eid, K_1, K_2) \in C$: compute $c$ as in steps 18, 19
24 :     Find $\mathsf{sk}_{\gamma,\delta} \in \mathcal{K}$ which decrypts the challenge $c$ to 1
25 :     Get $\pi \leftarrow \mathsf{NIZK.P}_{\mathsf{Dec}}(\mathsf{mpk}, c, (\gamma, \delta), \mathsf{sk}_{\gamma,\delta})$ and send $(\mathsf{claim}, eid, \pi, K_1, K_2, \gamma, \delta)$
26 :   Else broadcast an error message $(\mathsf{claim}, eid, \perp)$
27 : $(\mathsf{claim}, eid, \pi, K_1, K_2, \gamma, \delta)$ from $P_\nu$: compute $c$ as in steps 18, 19
28 :   If $(\nu, \gamma) \in R$ and $\pi$ is accepted: Send $(\mathsf{update}, \gamma, n)$ to $\mathcal{F}_{\mathsf{SK}}$
29 :     Update $S \leftarrow S \cup \{\gamma + \delta' n \ : \ \delta' \in [\kappa]\}$ and return $(\mathsf{result}, eid, \nu)$
30 :   Else: return $(\mathsf{rejected}, eid, \nu)$
```

Fig. 11. Protocol $P^{(i)}_{\mathsf{SSLE},\kappa}$. $\mathbf{S} \in \mathbb{G}_1^{7,4}$ represents the linear operations in lines 6-8.

such that the probability of Q containing only corrupted users is smaller than $2^{-\lambda}$. Notice that $t < N/2$ implies $\ell \leq \lambda$.

In steps 1-6 members of the committee sample a polynomial f_i used for the VSS, and shares $\mathbf{w}_{i,\alpha}$. The proof in line 4 guarantees that the adversary is aware of the plaintext $f_i(j)$ encrypted, preventing decryption-oracle attacks.

In lines 7-15 users test the VSS by checking if the exponents of $h_\mu, (k_{i,\mu})_{i=0}^{N-1}$ lies in the right Reed-Solomon code. A standard test is to check orthogonality with a codeword in the dual space $\mathsf{RS}^\perp_{\mathbb{F},N+1,t}$. Next, consistency with $s_{i,\mu} = f_\mu(i)$ and $k_{i,\mu}$ is checked. If it fails the player will complain (lines 10-13) and remove P_μ from the committee.

Next, the generation of a new secret key begins by querying $\mathcal{F}^{\mathsf{sk}}_{\mathsf{CT}}$, line 20, which returns $(d_\delta)_{\delta=0}^{k-1}$, interpreted as the randomness of requested OFE keys. In lines 21-25 members of the committee generate the secret key share $\mathbf{d}^{(i)}_{n,\delta}$ and privately send it to the receiver. Again a NIZK is added to prevent any decryption-oracle attack.

Observe now that, for every $\mu \in Q$

$$(\mathbf{z}_0, z_{1,\mu}, z_{2,\mu}, z_{2,\mu}) \;=\; [\mathbf{a}]_1\,,\; \left[\mathbf{a}^\top \mathbf{w}_{1,\mu}\right]_1,\; \left[\mathbf{a}^\top \mathbf{w}_{2,\mu}\right]_1,\; \left[\mathbf{a}^\top \mathbf{w}_{3,\mu}\right]_1$$

is a master public key for our OFE scheme, and $(\mathbf{d}^{(\mu)}_{n,\delta}, d_\delta)$ is a secret key for $(1, n, \delta)$ in the same scheme. Hence the recipient, lines 26-31, verifies this key share by checking if it is able to decrypt an encryption of $\mathbf{0}$. Somewhat surprisingly in the proof of security we show that this is enough to ensure correctness of the key.

Finally, if the above check fails, the recipient broadcasts a complaint message exposing the malformed key. Every user then checks the complaint and, if legitimate, remove P_μ from the committee.

Theorem 3 *Protocol $\{P^{(i)}_{\mathsf{SK}} \,:\, i \in [N]\}$ securely realises $\mathcal{F}_{\mathsf{SK}}$ in the $(\mathcal{F}_{\mathsf{CT}}, \mathcal{F}_{\mathsf{zk}})$ hybrid model under the* SXDH *assumption for the class of* PPT *environments $\mathcal{Z}$ that statically corrupt up to $\lfloor N/2 \rfloor$ players.*

6 Efficiency considerations

Overall communication costs of our protocol are summarised in Table 2. As mentioned in the previous section, however most of these messages are not required for verification and, in particular, they do not need to be stored on chain.

More in detail, for the VSS to generate the ElGamal public and secret keys, only aggregated elements $h, k_0, \ldots, k_{N-1}$ have to be placed on-chain, as those are the only ones required to verify the secret sharing. Next, during elections, we have to store the partial ciphertexts and related NIZKs sent by the committee, as these components are necessary to reconstruct the election's ciphertext. However, our specific OFE and protocol allow the winner to aggregate the expensive threshold decryption, without the need to also post a proof of correctness. Note that the

Party $P^{(i)}_{\mathsf{SK}}$ realising $\mathcal{F}_{\mathsf{SK}}$:

```
    Initially set $n \leftarrow 0$, $S \leftarrow \varnothing$. Create $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{AE.Setup}(1^\lambda)$, broadcast
    $(\mathsf{user\_key}, \mathsf{pk}_i)$, send $(\mathsf{toss}) \to \mathcal{F}^{\mathsf{stp}}_{\mathsf{CT}}$ and wait for its reply $(\mathsf{tossed}, Q, \mathbf{z}_0, g)$
1 : If $i \in Q$: Sample $f_i \xleftarrow{\$} \mathbb{F}_q[x]_{<t}$, $\mathbf{w}_{1,i}, \mathbf{w}_{2,i}, \mathbf{w}_{3,i} \xleftarrow{\$} \mathbb{F}_q^2$
2 :     Compute $h_i \leftarrow g^{f_i(-1)}$, $k_{j,i} \leftarrow g^{f_i(j)}$ and $z_{\alpha,i} \leftarrow \mathbf{z}_0^{\mathbf{w}_{\alpha,i}}$ for $j \in [N]$, $\alpha \in [3]$
3 :     $c_{j,i} \xleftarrow{\$} \mathsf{AE.Enc}(f_i(j), \mathsf{pk}_j)$ with randomness $r_{j,i}$
4 :     $\pi_{j,i} \xleftarrow{\$} \mathsf{NIZK.P}_{\mathsf{Enc}}(c_j, \mathsf{pk}_j, f_i(j), r_{j,i})$
5 :     Broadcast $(\mathsf{msg}, h_i, (k_{j,i}, c_{j,i}, \pi_{j,i})_{j=0}^{N-1})$
6 :     Send $(\mathsf{prove}, (z_{\alpha,i})_{\alpha=1}^3, (\mathbf{w}_{\alpha,i})_{\alpha=1}^3)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{Lin}}}$
7 : When $P_\mu \to (\mathsf{msg}, h_\mu, k_{j,\mu}, c_{j,\mu}, \pi_{j,\mu})_{j=0}^{N-1}$, $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{Lin}}} \to (\mathsf{proof}, \mu, z_{\alpha,\mu})_{\alpha=1}^3$ for $\mu \in Q_0$:
8 :     Set $\mathbf{k}_\mu = (h_\mu, k_{0,\mu}, \dots, k_{N-1,\mu})$ and sample $\mathbf{v} \xleftarrow{\$} \mathsf{RS}^\perp_{\mathbb{F},N+1,t}$.
9 :     If $\mathbf{k}_\mu^{\mathbf{v}} \neq 1$ or some $\pi_{j,\mu}$ is rejected: remove $\mu$ from $Q_0$.
10 :    Decrypt $s_{i,\mu} \leftarrow \mathsf{AE.Dec}(c_{i,\mu}, \mathsf{pk}_i, \mathsf{sk}_i)$. If $g^{s_{i,\mu}} \neq k_{i,\mu}$:
11 :        $\pi \leftarrow \mathsf{NIZK.P}_{\mathsf{Dec}}(\mathsf{pk}_i, c_{i,\mu}, s_{i,\mu}, \mathsf{sk}_i)$, and broadcast $(\mathsf{complain}, s_{i,\mu}, \mu, \pi)$
12 :    Upon receiving $(\mathsf{complain}, \mu, s_{i,\mu}, \pi)$ from $P_j$:
13 :        If $\pi$ is accepting and $g^{s_{i,\mu}} \neq k_{i,\mu}$, remove $\mu$ from $Q_0$.
14 :    Compute and store $z_\alpha \leftarrow \prod_{\mu \in Q_0} z_{\alpha,\mu}$, $h \leftarrow \prod_{\mu \in Q_0} h_\mu$, $k_j \leftarrow \prod_{\mu \in Q_0} k_{j,\mu}$
15 :    $\mathsf{mpk} \leftarrow (\mathbf{z}, z_1, z_2, z_3)$, $\mathsf{pp} \leftarrow (\mathsf{mpk}, h, k_0, \dots, k_{N-1})$, $s_i \leftarrow \prod_{\mu \in Q_0} s_{i,\mu}$
```

Fig. 12. Realisation $\mathcal{F}_{\mathsf{SK}}$, Initial setup phase

same property does not hold for the first round, since together with the partial ciphertexts one would have to aggregate the corresponding NIZKs with more sophisticated tools. Finally we remark that it is also possible to avoid storing encrypted secret keys for our OFE on chain, using the chain only for disputes.

As shown in the Table, while election requires low communications, the setup is more expensive, requiring 252 MB for 2^{14} users. However, this is supposed to be performed rarely. Once this is done, our protocol allows new users to join providing them a new secret key, without updating the key material of other users. This registration takes only 73 KB of communication. Letting users leave the system on the other hand introduces some inefficiencies. The problem is that users who go away may still be elected, causing some elections to end without a winner. An obvious, but expensive, way to completely remove this problem is to perform a new setup every time that one or more users leave. However, one can also make a trade-off leaving the possibility that some elections finish without a winner, and redo the setup only when this probability (which for L inactive users out of N registered users is L/N) becomes too high.

Comparison with [BEHG20] We now compare our UC-secure construction with the shuffle-based solution in [BEHG20], which we briefly recall here. Essentially the public state contains a list of Diffie-Hellman pairs $(K_{i,1}, K_{i,2})$, one

Party $P_{\mathsf{SK}}^{(i)}$ realising $\mathcal{F}_{\mathsf{SK}}$ upon receiving:

```
17 : $(\mathsf{setup})$: Return $(\mathsf{input}, \mathsf{pp}, s_i)$
18 : $(\mathsf{update}, n, \gamma)$: set $S \leftarrow \{\gamma + \delta n \ : \ \delta \in [\kappa]\}$
19 : $(\mathsf{keygen})$: Broadcast $(\mathsf{key\_request})$
20 : $(\mathsf{key\_request})$ from $P_j$: Send $(\mathsf{toss}, rid|j)$ to $\mathcal{F}_{\mathsf{CT}}^{\mathsf{sk}}$ and return $(\mathsf{key\_requested}, j)$
21 : $(\mathsf{tossed}, rid|j, (d_\delta)_{\delta=0}^{\kappa-1})$ from $\mathcal{F}_{\mathsf{CT}}^{\mathsf{sk}}$, if $i \in Q$:
22 :   While $n \in S$, increase $n \leftarrow n + 1$
23 :   $\mathbf{d}_{n,\delta}^{(i)} \leftarrow [\mathbf{w}_{1,i} + n\mathbf{w}_{2,i} + \delta\mathbf{w}_{3,i}]_{d_\delta}$ ,   $\mathbf{d}_n^{(i)} \leftarrow (\mathbf{d}_{n,\delta}^{(i)})_{\delta=0}^{\kappa-1}$
24 :   $c_i \xleftarrow{\$} \mathsf{AE.Enc}(\mathbf{d}_n^{(i)}, \mathsf{pk}_j)$ with randomness $r_i$
25 :   $\pi_i \xleftarrow{\$} \mathsf{NIZK.P}_{\mathsf{Enc}}(c_i, \mathsf{pk}_j, \mathbf{d}_n^{(i)}, r_i)$; Broadcast $(\mathsf{key\_partial}, c_i, \pi_i, j, n)$
26 : $(\mathsf{key\_partial}, c_\mu, \pi_\mu, i, n)$ with accepting $\pi_\mu$ from $P_\mu$ for $\mu \in Q_n$:
27 :   for all $\mu \in Q_n$ get $(\mathbf{d}_{n,\delta}^{(\mu)})_{\delta=0}^{\kappa-1} \leftarrow \mathsf{AE.Dec}(c_i, \mathsf{sk}_i)$
28 :   If $e(\mathbf{z}_0, \mathbf{d}_{n,\delta}^{(\mu)}) \neq e(z_{1,\mu} \cdot z_{2,\mu}^n \cdot z_{3,\mu}^\delta, d_\delta)$:
29 :     Remove $\mu$ from $Q$ and compute $\pi$ a proof that $c_\mu$ encrypts $(\mathbf{d}_{n,\delta}^{(\mu)})_{\delta=0}^{\kappa-1}$
30 :     Broadcast $(\mathsf{key\_complain}, \mu, n, \delta, (\mathbf{d}_{n,\delta}^{(\mu)})_{\delta=0}^{\kappa-1}, \pi)$
31 :   Set $\mathsf{sk}_{n,\delta} \leftarrow \left(\prod_{\mu \in Q_n} \mathbf{d}_{n,\delta}^{(\mu)}, \, d_\delta\right)$ and return $(\mathsf{key}, (\mathsf{sk}_{n,\delta})_{\delta=0}^{\kappa-1})$
32 : $(\mathsf{key\_complain}, \mu, n, \delta, (\mathsf{sk}_{n,\delta}^{(\mu)})_{\delta=0}^{\kappa-1}, \pi)$ from $P_j$ with accepting $\pi$:
33 :   Perform the test on line 28. If the two terms differ:
34 :     Remove $\mu$ from $Q_n$, and $P_\mu$'s share from $\mathsf{mpk}$
35 :     For each key received $\mathsf{sk}$ let $\mathbf{d}_\mu$ be $P_\mu$'s share
36 :     Parse $\mathsf{sk} = (\mathbf{d}, d)$, return $(\mathsf{key\_update}, (\mathbf{d} \cdot \mathbf{d}_\mu^{-1}, d))$
```

Fig. 13. Realisation $\mathcal{F}_{\mathsf{SK}}$, Key Distribution phase

for every user, and P_i's secret key is a discrete log k_i such that $K_{i,2} = K_{i,1}^{k_i}$. An election is performed by choosing one of those tuples through the random beacon and the leader claims victory by revealing its secret key. To achieve unpredictability, each time a pair is added by a user, he sends a shuffled and re-randomized list along with a NIZK. Note that every election involves at least the registration of the previous winner, who has "burnt" her secret key, if she desires to stay. Moreover, this implies that the protocol requires at each round as many shuffles as the number of new users. Notably, all the lists and NIZKs have to be posted on chain in order to ensure verifiability.

In the high communication solution, denoted N-shuffle, each shuffles costs $2n$ group elements, while the more efficient and less secure one, denoted $\sqrt{N}$-shuffle, costs $2\sqrt{n}$ elements.

Procedure	**Number of elements sent**			**Size**	
	$\mathbb{F}_q$	$\mathbb{G}_1$	$\mathbb{G}_2$	*off-chain*	*on-chain*
VSS for ElGamal	$2\lambda N$	$2\lambda N + 2\lambda$	–	252 MB	1.05 MB
Distribute mpk	3λ	2λ	–	20.5 KB	20.5 KB
Election, 1st Round	$\kappa(6 + 2\log n)$	$\kappa(7 + \log n)$	–	34.0 KB	34.0 KB
Election, 2nd Round	$2(t+1)$	$2(t+1)$	–	1.57 MB	–
Election, Claim	1	2	3	256 B	256 B
Registration	λ	–	$2\kappa\lambda + 2\lambda$	73.3 KB	–

Table 2. Communication costs of our scheme, using ElGamal in place of the generic IND-CPA encryption. Size is computed assuming $\log|\mathbb{F}_q| = 256$, $\log|\mathbb{G}_1| = 512$, $\log|\mathbb{G}_2| = 256$, $\log|\mathbb{G}_T| = 3072$, $\lambda = 80$, $\kappa = \log N$, $t = \lfloor N/2 \rfloor$ and $N = 2^{14}$.

In light of the requirement in [Lab19] to support $O(\log^2 N)$ new users per round, we compare these solutions evaluating the cumulative cost of several elections, interleaving between every two a fixed amount of registrations. In Fig. 6 we provide the communication costs for such a scenario where we assume to start with 2^{14} users and then perform: 10 registrations for each election in the first column, 20 in the second column, and 30 in the third one. Furthermore we let the same number of new users who joined the system leave it after each election. Note that, as mentioned earlier, this means some elections may have to be repeated in our case as users who leave may still be elected.

We remark that in those plots, the costs of the shuffle-based solutions do not even include the costs of setup[15], as it can be done only once in contrast to ours where we need to occasionally refresh the secret key material. In spite of that, the cost of our setup is quickly compensated by our lighter registration and election procedure, which makes our solution more suited to dynamic scenarios.

More efficient SSLE with Game Based Security We now remark that communication complexity can be further reduced in our construction at the cost of giving up UC security yet achieving the game-based security notion.

As we would not need any more to simulate each election, every secret key can now be produced without artificially skipping some of them. For the same reason, the NIZKs need not to be simulation-extractable, which allow us to use Bulletproofs for the range proofs. This reduces on-chain costs to $O(\kappa \log\log N)$.

Finally, when giving up UC security users who voluntarily leave the system can be handled by asking such users to reveal their own secret keys upon leaving, as done in [BEHG20]. This way, if a revoked user happens to be elected, everyone can detect it and immediately proceed to generate a new election's ciphertext. To keep round complexity low, one can also prepare several challenges per election,

[15] I.e. the cost to generate a shuffled list containing the pairs of the initial users. This has cost $O(n^2)$ if everyone performs a shuffle, or $O(\kappa n)$ using an approach similar to ours where a random committee of κ users shuffle the initial list

order them, remove those that can be decrypted with keys of users who left, and set the current challenge as the first of the remaining ones. This solution only works for non-UC security though, as the simulator should now generate on request honest user's secret key that are consistent with previous elections.

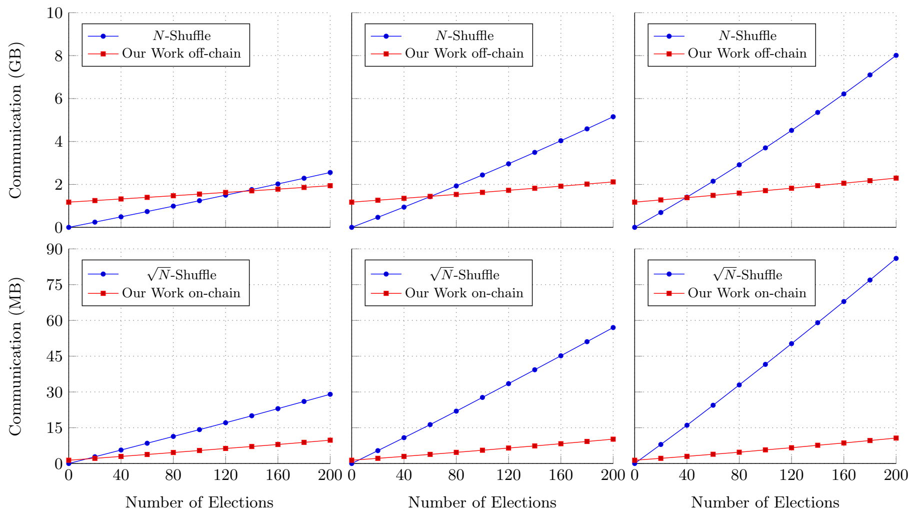

Fig. 14. Cumulative communication costs in this work and [BEHG20]. Initially the number of users is $N = 2^{14}$ and between every two elections 10 (left column), 20 (middle column) or 30 (right column) registrations occur, while the same amount of already registered users leave.

Acknowledgements

This work has received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by a research grant from Nomadic Labs and the Tezos Foundation, by the Programma ricerca di ateneo UNICT 2020-22 linea 2, by SECURING Project (PID2019-110873RJ-I00/MCIN/AEI/10.13039/501100011033), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), CRYPTOEPIC (ref. UR2019-103816), RED2018-102321-T, and PRODIGY (TED2021-132464B-I00), and by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339). The last five projects are co-funded by European Union EIE, and NextGenerationEU/PRTR funds.

References

ABC$^+$05. M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, Heidelberg, August 2005.

AC21. S. Azouvi and D. Cappelletti. Private attacks in longest chain proof-of-stake protocols with single secret leader elections. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 170–182, 2021.

AMM18. S. Azouvi, P. McCorry, and S. Meiklejohn. Betting on Blockchain Consensus with Fantomette. *CoRR*, abs/1805.06786, 2018.

BBB$^+$18. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BDOP04. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, Heidelberg, May 2004.

BEHG20. D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24, 2020.

BGG$^+$18. D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

BGM16. I. Bentov, A. Gabizon, and A. Mizrahi. Cryptocurrencies Without Proof of Work. In J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. S. Wallach, M. Brenner, and K. Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 142–157. Springer, Heidelberg, February 2016.

BPS16. I. Bentov, R. Pass, and E. Shi. Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919, 2016. `http://eprint.iacr.org/2016/919`.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional Encryption: Definitions and Challenges. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

Can01. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CD20. I. Cascudo and B. David. ALBATROSS: Publicly AttestabLe BATched Randomness Based On Secret Sharing. In S. Moriai and H. Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 311–341. Springer, Heidelberg, December 2020.

CDG$^+$18. J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The Wonderful World of Global Random Oracles. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018.

CF01. R. Canetti and M. Fischlin. Universally Composable Commitments. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

CJS14. R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a Global Random Oracle. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014.

DKT$^+$20. A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni. Everything is a Race and Nakamoto Always Wins. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 20*, pages 859–878. ACM Press, November 2020.

FKMV12. S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the Non-malleability of the Fiat-Shamir Transform. In S. D. Galbraith and M. Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012.

FLS90. U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

GGH$^+$13. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

GHM$^+$17. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68, New York, NY, USA, 2017. Association for Computing Machinery.

GOP$^+$21. C. Ganesh, C. Orlandi, M. Pancholi, A. Takahashi, and D. Tschudi. Fiat–Shamir Bulletproofs are Non-Malleable (in the Algebraic Group Model). Cryptology ePrint Archive, Paper 2021/1393, 2021. `https://eprint.iacr.org/2021/1393`.

GOT19. C. Ganesh, C. Orlandi, and D. Tschudi. Proof-of-Stake Protocols for Privacy-Aware Blockchains. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 690–719. Springer, Heidelberg, May 2019.

GPS08. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for Cryptographers. *Discrete Appl. Math.*, 156(16):3113–3121, September 2008.

KKKZ19. T. Kerber, A. Kiayias, M. Kohlweiss, and V. Zikas. Ouroboros Crypsinous: Privacy-Preserving Proof-of-Stake. In *2019 IEEE Symposium on Security and Privacy*, pages 157–174. IEEE Computer Society Press, May 2019.

KMTZ13. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally Composable Synchronous Computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

KSW08. J. Katz, A. Sahai, and B. Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, Heidelberg, April 2008.

Lab19. P. Labs. Secret single-leader election (SSLE). `https://web.archive.org/web/20191228170149/https://github.com/protocol/research-RFPs/blob/master/RFPs/rfp-6-SSLE.md`, 2019.

Mau15. U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *Designs, Codes and Cryptography*, 77(2):663–676, 2015.

NR97. M. Naor and O. Reingold. Number-theoretic Constructions of Efficient Pseudo-random Functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

O'N10. A. O'Neill. Definitional Issues in Functional Encryption. Cryptology ePrint Archive, Report 2010/556, 2010. `http://eprint.iacr.org/2010/556`.

Sah99. A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

Wee17. H. Wee. Attribute-Hiding Predicate Encryption in Bilinear Groups, Revisited. In Y. Kalai and L. Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 206–233. Springer, Heidelberg, November 2017.