Succinct Publicly-Certifiable Proofs

or, Can a Blockchain Verify a Designated-Verifier Proof?

Matteo Campanelli and Hamidreza Khoshakhlagh

Aarhus University, Denmark
{matteo,hamidreza}@cs.au.dk

Abstract. We study zero-knowledge arguments where proofs are: of knowledge, short, publicly-verifiable and produced without interaction. While zkSNARKs satisfy these requirements, we build such proofs in a constrained theoretical setting: in the standard-model—i.e., without a random oracle—and without assuming public-verifiable SNARKs (or even NIZKs, for some of our constructions) or primitives currently known to imply them.

We model and construct a new primitive, SPuC (Succinct Publicly-Certifiable System), where: a party can prove knowledge of a witness w by publishing a proof π_0; the latter can then be certified non-interactively by a committee sharing a secret; any party in the system can now verify the proof through its certificates; the total communication complexity should be sublinear in $|\mathsf{w}|$. We construct SPuCs *generally* from (leveled) FHE, homomorphic signatures and linear-only encryption, all instantiatable from lattices and thus plausibly quantum-resistant. We also construct them in the two-party case replacing FHE with the simpler primitive of homomorphic secret-sharing.

Our model has practical applications in blockchains and in other protocols where there exist committees sharing a secret and it is necessary for parties to prove knowledge of a solution to some puzzle. Our constructions can be seen as a way to compile a designated-verifier SNARK into a proof system with a flavor of public-verifiability with similar efficiency features of the starting dvSNARK (e.g., proving time).

We show that one can construct a version of SPuCs with robust proactive security from similar assumptions. In a proactively secure model the committee reshares its secret from time to time. Such a model is robust if the committee members can prove they performed this resharing step correctly. Along the way to our goal we define and build Proactive Universal Thresholdizers, a proactive version of the Universal Thresholdizer defined in Boneh et al. [Crypto 2018].

Table of Contents

1 Introduction

We consider the setting where, at any given moment in time, users can post a puzzle on a blockchain. Later some other user may come along and show to everybody that they know a solution to the puzzle without necessarily leaking it (i.e., in zero-knowledge). This scenario has numerous applications to problems in secure decentralized computing that have received much attention lately—these include but are not limited to: showing that the parties are following some internal protocol [31], storing and retrieving secrets with functionalities close to extractable witness encryption [33], zero-knowledge contingent payments [19] and showing knowledge of secret inputs to general smart-contracts [17].

For a solution to the problem above to be useful, we do not only require that all the users can verify the proof to a puzzle, but we also need to pose efficiency requirements. A *scalable* solution should involve minimal interaction among parties —ideally the *puzzle-solver* should post its proof and then disappear—and low bandwidth—short proofs.

In principle, a perfect candidate for this setting are publicly-verifiable succinct non-interactive arguments of knowledge (or pv-SNARKs) [4] with zero-knowledge properties. In this work, however, we shall seek solutions that do not require publicly-verifiable SNARKs. *Our choice is motivated by exploring different (and, plausibly, weaker) assumptions while obtaining post-quantum secure constructions.* In our solutions we do not only avoid using pv-SNARKs, but also any *publicly-verifiable proof for non-deterministic computations* (that is, NIZKs for $\mathsf{NP} \setminus \mathsf{P}$). We discuss the rationale of this choice in Section 1.2.

Committees certifying proofs without interaction. We consider a model which is almost as non-interactive as that of pv-SNARKs, but in which we add one more hop. At the high-level our model works as follows. At each moment in time there exists a committee holding a secret (the secret being shared among the committee members)[1].This secret permits them to publicly "certify" a proof publicly posted by anybody claiming they know a solution to a puzzle. Certifying a proof happens in a threshold fashion: a prover holding a witness w, outputs a succinct proof π_0; the parties in the committee can then process it broadcasting a "partial certificate", which any node in the network can check whether to consider it valid; if at least d (out of the total N) committee members broadcasted a valid certificate, these can be combined through a deterministic algorithm to obtain the bit b determining acceptance/rejection of π_0. The protocol is required to stay secure as long as the adversary corrupts less than a certain fraction of committee members.

Naturally, a general MPC-based solution is always applicable in this scenario. Our challenge, however, is to keep the efficiency requirements of low interaction/bandwidth sketched above. Specifically we need to guarantee that: *(i)* parties require no interaction among each other for proving, certification or ver-

[1] Such a committee is not an uncommon architectural choice. See, e.g. [3, 29, 18]

ification[2]; *(ii)* all messages—the proof π_0 as well as the partial certificates—are short (sublinear in the witness size).

YOSO-style proactive committees The requirements sketched above are sufficient for the setting where a certifying committee is *static* (this is our vanilla SPuC model in Section 4). We also study a version of our protocol where the committee changes over time and can proactively reshare its shares. The challenge for us is to make these protocols *robust* and *YOSO-style*, staying within our weak-assumptions framework as much as possible. Requiring robustness means that the resharing parties can prove whether they reshared correctly. YOSO (You Only Speak Once) [29] requires more elaboration: when performing the resharing (as well as in other parts of the protocol) parties should not interact among each other, instead they speak only once and then can potentially disappear. For us, the YOSO-style requirement means that, after the parties have been assigned their roles as members of the certifying committees[3] they need to speak only once. Their message will consist of the certificate for the (potentially many) proofs π_0-s publicly posted during their time holding the role.

1.1 Contributions and Overview

1.1.1 A Model for SPuCs We provide a formal model for Succinct Publicly Certifiable proofs (SPuCs), which we describe in Section 4. Our security notions all refer to an adversary controlling up to $\mathsf{d}-1$ of the N committee members. We require properties analogous to those for proof systems: *unbounded zero-knowledge*—an adversary cannot learn anything even after (adaptively) querying many proofs and certificates on them—and *strong knowledge-soundness*—given an adversary providing a verifying proof and certificates for a statement stmt one can extract a valid witness from them. The last notion can be paraphrased as: no adversary can forge a certified proof π_0 and (up to $\mathsf{d}-1$) valid certificates for stmt and π_0 without knowing a witness for stmt. While these definitions intuitively are extensions of the corresponding notions for designated-verifier NIZKs, we find them to be non-trivial and require some care (for example, in modeling appropriate oracles for the zero-knowledge simulator). Finally we require our proofs and certificates to be of total size sublinear in the witness size.

1.1.2 A General Construction for SPuCs We provide a general construction for SPuCs from designated-verifier SNARKs and a primitive called Universal Thresholdizers (UT) introduced in [9]. Informally a UT generalizes threshold primitives such as threshold encryption or signatures. The setup of a UT takes as

[2] Naturally we require certifying parties to wait for proof π_0 to be posted publicly.

[3] This happens through some nomination mechanism that we just posit and do not model explicitly in this paper. For example, one could use the nominating committee techniques in [3]. After being nominated the committee members can potentially remain anonymous to the rest of the network. This can be done for example through ephemeral public-keys and anonymous public-key encryption [3]

input a secret x and produces some public parameters and N secret keys which, in our setting, will be given to the members of the certifying committee. These allow the secret-holders to non-interactively and jointly compute any circuit C on the secret without knowing the secret. Each of these "local computations" from the secret holders can be verified as being valid. If at least d of them are valid they can be recombined to reconstruct $C(x)$.

Our second ingredient are designated-verifier (dv) SNARKs. In a dvSNARK a proof π for a statement stmt can be verified only by a party holding a verification-key vk through $\mathsf{Verify}(\mathsf{vk}, \mathsf{stmt}, \pi)$. To preserve soundness of the system it is important that the designated-verifier key remains secret from a malicious prover.

We show that thresholdizers from [9] can be used to construct SPuCs by injecting $x = \mathsf{vk}$ as a secret in the UT and compute functions of the type $C_{\mathsf{stmt},\pi}(\cdot) = \mathsf{Verify}(\cdot, \mathsf{stmt}, \pi)$ through it. Although this construction is arguably simple, showing we can apply UT to obtain our desired argument-like system has some non-trivial aspects to it. First, despite the generality of the threshold-like setting in the UT definition, its security definition is incompatible with that in the SPuC setting: the latter involves additional oracles—e.g. a proof oracle for true statements in the zero-knowledge experiment—and additional experiments extractability. Second, we do not require the "full universality" of these thresholdizers, but only that their supported computations include dvSNARK verification. This may be a low-complexity computation, involving for example a decryption and a zero-test on a low-degree polynomial [6]. This is significantly less complex than the proven relation $\mathcal{R}$. Finally, to obtain zero-knowledge in SPuCs we observe that the dvSNARK used in the construction does not need to satisfy the usual notion of zero-knowledge for designated-verifier NIZKs; a weaker notion suffices. We introduce and model this notion, dubbed "key-less zero-knowledge", which we believe to be of independent interest. We show that some existing dvSNARKs already satisfy this notion, namely all those obtained through the popular compiler from Non-Interactive Linear Proofs (NILPs) [34] described in [6]. We also observe that it is possible to obtain dvSNARKs satisfying this notion by compiling a (non zero-knowledge) pvSNARKs with a public-key encryption scheme.

1.1.3 Quantum-Resistant Instantiations from Homomorphic Primitives and Linear-Only Encryption

By "opening the boxes" of UT and dvSNARKs we show we can instantiate our SPuC construction requiring the existence of: (leveled) Fully Homomorphic Encryption (FHE) [27][4], Homomorphic Signatures (HS) with context-hiding[5] properties [32] and linear-only encryption [6, 12] (we require the existence of *all* these primitives). Given an encryption of x, FHE allows computing an encryption of $f(x)$ using only public parameters; *leveled* FHE ensures correctness only for functions of a bounded depth d spec-

[4] More precisely, we require leveled Threshold FHE, which is shown to be implied by leveled FHE with the mild requirement of moderate decryption "noise bound"[9].

[5] Context-hiding states that a signature $\sigma_{f,x}$, authenticating $f(x)$ and obtained homomorphically from a signature on x, reveals nothing about x.

ified at setup time. HS allows to perform the same on signatures. Linear-only encryption [6, 12] is a form of linearly-homomorphic encryption with guaranteed limited malleability.

We elaborate more on the relation between publicly-verifiable NIZKs and these abstract primitives in Section 1.2. Our construction for SPuCs is quantum-resistant: all the above primitives can be instantiated from lattices. This is of particular relevance since there are no results on publicly-verifiable zero-knowledge arguments with short proofs in the standard model. The only other construction of pvNIZKs from lattices does not have succinct proofs [37]; the constructions in [12, 13, 26] are for designated-verifiers.

One relatively minor challenge for us here is making sure that UT can be built through the abstract primitives above. The construction for UT in [9] is based on NIZKs, which we want to avoid. Although [9] informally mentions that one could replace NIZKs with context-hiding homomorphic signatures, there is no formal construction in the paper[6].

We also show yet one more construction of UT for the two-party case replacing FHE with the simpler notion of (two-party) homomorphic secret sharing (HSS), which can be also built from lattices [16]. An HSS scheme allows to share a secret x and to let the share-holders compute shares of $C(x)$ for any circuit C.

A practical perspective: If applied to an efficient dvSNARK—potentially one more efficient than pvSNARKs—then one could leverage our constructions to obtain a public-verifiability-flavored proof system preserving some of the efficiency features of the starting dvSNARK (e.g. proving time and to some extent succinctness). We believe our constructions can be practical: their overhead for certifying dvSNARK proofs is arguably low since we apply homomorphic cryptography to very small circuits (those for dvSNARK verification) and so is the communication overhead for each certificate (whose number, however, scales with the chosen threshold). Nonetheless it seems unclear what scheme could *currently* be used for such instantiation. To the best of our knowledge, despite recent advances [35] it is still an open problem to obtain dvSNARKs without random oracles that could beat pvSNARKs in practice (especially for prover's time).

1.1.4 Proactive SPuCs from Proactive UT (pUT)

We consider the setting where the committee is not fixed but it can proactively reshare its secret at every round without interaction. We construct a proactive variant of SPuCs through a proactive variant of UT (pUT) which we introduce in this paper. The construction of pSPuC from pUT is analog to that SPuC from UT, i.e. applying a thresholdizer for a designated-verifier SNARK. Our model for proactive SPuCs is straightforward once defining SPuCs and pUT.

A Construction of pUT from Special UT (sUT). To convert a UT into its proactive version we need to enable the committee members to reshare (or "hand

[6] A formal construction from homomorphic signatures is present in [10] but it relies on the specifics of the underlying homomorphic encryption scheme.

over") their secret keys. Some prior techniques for doing this allow each party to prove they are resharing correctly but they use NIZKs [3, 33]. For our alternative approach, we observe that what needs to be handed over to the next committee are reshares of some trapdoor computed inside the UT setup. We cannot directly perform secret shares of it at handover time because there is no party holding it (whoever computed it is now disappeared or it was not a single entity but a protocol execution).

The solution to the problem above comes from UT itself. UT can be used to verifiably compute functions on some secret obliviously (without knowing the secret). Can we then extend UT to perform oblivious computation, not only an injected secret x, but on its own trapdoor (a secret computed during the execution of $\mathsf{UT.Setup}(x)$)? With this tool in our hands we could then let the committee members obliviously compute some resharing function $\mathsf{Reshare}$.

We show we can extend UT to support general (controlled) evaluations of its own trapdoor obtaining a new primitive we call sUT. A sUT is like a UT but it allows evaluations on two secrets: some secret x specified (through algorithm $\mathsf{sUT.PartEval}$) at setup time and its own trapdoor (through an analog of the partial evaluation algorithm, called $\mathsf{sUT.TrapEval}$). We are able to construct sUT using almost the same assumptions as for UT: we still require homomorphic signatures and FHE, but we need to also assume circular security of the latter (namely, we should be able to securely encrypt its own decryption key in it).

We then show how to construct pUT by applying the algorithm $\mathsf{TrapEval}$ of sUT on a function that generates a new secret and provides its share. Other techniques of our construction for pUT are inspired by the YOSO-style ones in [3] where the committee members of the new epoch can access their share by opening a ciphertext encrypted with an ephemeral public key (of which they only know the decryption key).

Both pUT and sUT are of independent interest and can be applied in contexts of "cryptography-as-a-service" as those described in [3].

$$\begin{tikzcd}
\text{lin-only enc.} \arrow[dr, "{[6]}"] & \text{TFHE} \arrow[dr] & & \text{sUT} \arrow[r] & \text{pUT} \arrow[d] \\
\text{sub-exp SIS} \arrow[dr, "{[32]}"] & \text{dvSNARK} \arrow[r] \arrow[dr] & \text{UT} \arrow[r] \arrow[ur, dashed] & \text{SPuC} & \text{pSPuC} \\
& \text{HS} \arrow[ur] \arrow[r] & \text{UT}_{(2,2)} \arrow[r] & \text{SPuC}_{(2,2)} & \\
\text{LWE} \arrow[uuur, "{[9]}"] \arrow[r, "{[16]}"] & \text{HSS}_{(2,2)} \arrow[ur] & & &
\end{tikzcd}$$

Fig. 1: Dependency diagram of assumptions and constructions. Suffix "$(2, 2)$" denotes two-parties. The dashed line refers to non blackbox constructions and additionally requires circular (KDM) security for the TFHE scheme. Plausible constructions for linear-only encryption can be instantiated from LWE [12].

1.2 Further Theoretical Motivation and Assumptions

Our goal is to build SPuCs through assumptions that are weaker than publicly-verifiable zero-knowledge SNARKs and NIZKs. Also, our goal is to stay in the standard model (without random oracles). In the next paragraphs we discuss some of the motivation behind this and how our constructions relate to these goals. Our hope is that this work can provide a new lens on constructions relying on publicly-verifiable proof systems.
Through this work, we want to make the following observation:

> In the standard model, we can obtain robust systems for succinct zero-knowledge proofs, without interaction among the prover and verifier even if publicly-verifiable NIZKs or SNARKs do not exist. It is possible to extend these results to the case of proactive resharing without relying on publicly-verifiable SNARKs.

Why not using publicly-verifiable SNARKs? We know that succinct arguments in general require non-falsifiable assumptions (in case of black-box reductions) [30]. Constructions of publicly-verifiable SNARKs usually go around this by: the sometimes problematic Fiat-Shamir in the random-oracle model; knowledge-of-exponent-like assumptions or idealized settings such as the generic or algebraic group models (e.g., [34, 1, 24]). Since our constructions use results implying designated-verifier SNARKs, we cannot get around the result in [30] and will have underlying non-falsifiable assumption (those required for linear-only encryption[6, 12]). Nonetheless we remark that our results still hold without a random oracle and in the following paragraphs we argue that there is still an advantage in moving from pvSNARKs to dvSNARKs as an assumption.

Assumptions in dvSNARKs vs in pvSNARKs. We observe it is plausible that dvSNARKs may require strictly weaker assumptions than pvSNARKs. In fact, we know that publicly-verifiable SNARKs are not a stronger primitive than designated-verifier SNARKs since we can always construct the latter from pvSNARK by encrypting the proof under the verifier's public key. We still do not know whether there is a theoretical separation between these two notions though.

Even if dvSNARKs were not strictly weaker than pvSNARKs as a primitive in the standard model, we might still obtain them from *different* (and, potentially, more plausible) assumptions. For example, consider the pvSNARK constructions in [25] and the dvSNARK constructions in [12, 13, 26]. Although all non-falsifiable, the mathematical objects they refer to are quite different (respectively, groups with bilinear pairings and lattices). We point that there also exist other constructions in the standard model such as the publicly verifiable arguments in [38], but they are based on the indistinguishability obfuscation, which is yet not standard.

Finally, SPuCs and their constructions relying on dvSNARKs, can be motivated by post-quantum security. We observe that, to the best of our knowledge, *there are no known constructions for pvSNARKs in the standard-model that are resistant to quantum attacks.* In fact, only recently the community learned about the possibility of post-quantum non-interactive zero-knowledge (with non-succinct proofs) [37].

On not requiring publicly-verifiable NIZKs in general. Our constructions for SPuCs not only do not require publicly-verifiable SNARKs, but they do not require publicly-verifiable NIZKs in general either. None of the assumptions we rely on—linear-only encryption, leveled FHE (or two-party HSS) and (context-hiding) homomorphic signatures—are known to imply pvNIZKs[7]. We observe that homomorphic signatures with the context-hiding property can be seen as a variant of non-interactive zero-knowledge (with short proofs) for *deterministic* computations on authenticated data. However, they do not allow to prove anything on general non-deterministic computations since witnesses can possibly be unauthenticated. The reason our work can afford this is that we use a trusted setup that "bootstraps" the system creating secret keys for authentication (homomorphic signatures) and threshold homomorphic decryption and signing the initial set of shares. After this step no party is assumed to have access to these secrets. We remark that it is possible to replace the trusted setup with an MPC execution. This, at the same time, shows a limitation of our work: for this MPC to run efficiently one would probably require publicly-verifiable NIZKs (interactive approaches should also be possible though). We leave alternative approaches to the latter as future work.

On assuming homomorphic cryptography for small computations. We remark that although we often express our assumptions as *general* "Fully" Homomorphic Encryption and (context-hiding) Homomorphic Signature in general, our requirements are actually weaker. We only need homomorphic properties on computations as decryption, PRFs and the final low-degree test of some designated-verifier SNARKs[6]. These are all computable in the class NC^1 [2].

1.3 Other Related Work

The work in [5] also discusses how to compose (unleveled) FHE and "proofs" of the verification algorithms to obtain succinct arguments of knowledge (in their Section 9). The differences between their work and ours is that we use a primitive that checks only deterministic computation (we use homomorphic signatures; they use NIZK arguments of knowledge) and that their construction cannot achieve public-verifiability from designated-verifiability. On the other hand, to

[7] Unleveled FHE—where homomorphic operations work correctly for any polynomial-size function $f(x)$ without any depth bound—does imply *designated-verifier* NIZKs [22]. The recent work in [20] shows, however, that circular (KDM-secure) unleveled FHE even implies pvNIZKs. For our proactive extensions, we assume KDM-secure *leveled* FHE for NC^1 which is known to imply (circular-secure) unleveled FHE through bootstrapping [27]. We observe, however, that while the assumptions in our proactive constructions are sufficient to imply pvNIZKs, they do not require the standard FHE bootstrapping, significantly improving the efficiency of homomorphic operations. Finally, circular-secure leveled FHE is not known to imply pvSNARKs.

obtain the latter, we work in a slightly different security model and we add one "hop" in the protocol.

The work in [28] investigates approaches to minimizing proof size. Their work requires a slightly different primitive called fully homomorphic hybrid encryption, and differently from us, requires it to support any computation (whereas we require that only for the circuit of the verifier in a designated-verifier SNARK).

Notation and Basic Background For any positive integer n, $[n]$ denotes the set $\{1, \ldots, n\}$. We denote vectors in boldface. We use the notation $O_\lambda(f(n))$ to denote $O(p(\lambda)f(n))$ where p is some polynomial in the security parameter. We consider all adversaries to be stateful.

2 Background on Designated-Verifier SNARKs

A dvSNARK has a key-generation algorithm $\mathsf{dvKeyGen}$ which returns an evaluation key ek and a verification key vk for an NP relation $\mathcal{R}$. The prover $\mathcal{P}_{\mathsf{SNARK}}$ takes in ek, a statement stmt and a witness w, and outputs a proof π, which can be verified through algorithm $\mathcal{V}_{\mathsf{SNARK}}$ taking as input (stmt, π), and the (secret) vk. Key properties of a dvSNARKs are: *Succinctness* (its proofs are short), *Knowledge-soundness* (we can extract a valid witness from a verifying proof), *Zero-knowledge* (a proof does not reveal anything more than the truth of the statement). There are constructions of dvSNARKs from linear-only encryptions [6, 12] (which can be plausibly instantiated from LWE).

Definition 1. *(Designated-Verifier Succinct Non-Interactive Argument of Knowledge) Let $\mathcal{C} = \{\mathcal{C}_\ell\}_{\ell \in \mathbb{N}}$ be a family of arithmetic circuits. Let $\mathcal{R}_\mathcal{C}$ be the corresponding circuit satisfiability relation with the associated language $\mathcal{L}_\mathcal{C}$. A succinct non-interactive argument of knowledge (SNARK) for $\mathcal{R}_\mathcal{C}$ with completeness error $c(\lambda)$ and soundness error $\epsilon(\lambda)$ is a triple $\Pi_{\mathsf{SNARK}} = (\mathsf{dvKeyGen}, \mathcal{P}_{\mathsf{SNARK}}, \mathcal{V}_{\mathsf{SNARK}})$ defined as follows:*

– $\mathsf{dvKeyGen}(1^\lambda) \to (\mathsf{ek}, \mathsf{vk})$: *The setup algorithm inputs the security parameter λ and returns an evaluation key* ek *and verification state* vk.
– $\mathcal{P}_{\mathsf{SNARK}}(\mathsf{ek}, \mathsf{stmt}, \mathsf{w}) \to \pi$: *The prove algorithm inputs a common reference string* ek*, a statement* stmt *and a witness* w*, and returns a proof π.*
– $\mathcal{V}_{\mathsf{SNARK}}(\mathsf{vk}, \mathsf{stmt}, \pi)$: *The verification algorithm inputs the verification state* vk*, a statement* stmt *and a proof π, and returns a bit $b \in \{0, 1\}$ indicating accept or reject.*

We say a Π_{SNARK} is designated-verifier (dv) if $\mathsf{vk} \neq \bot$ and only the owner of vk can verify proofs. In this work, we only focus on dvSNARKs.

A SNARK is required to satisfy the following properties:

– **Completeness.** For all $\lambda \in \mathbb{N}$ and all $(\mathsf{stmt}, \mathsf{w}) \in \mathcal{R}_\mathcal{C}$,

$$\Pr\left[\begin{matrix}(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{dvKeyGen}(1^\lambda) \\ \pi \leftarrow \mathcal{P}_{\mathsf{SNARK}}(\mathsf{ek}, \mathsf{stmt}, \mathsf{w})\end{matrix} : \mathcal{V}_{\mathsf{SNARK}}(\mathsf{vk}, \mathsf{stmt}, \pi) = 1\right] \geq 1 - c(\lambda)$$

We say that Π_{SNARK} satisfies statistical completeness if $c(\lambda) = \mathsf{negl}(\lambda)$ and perfect completeness if $c(\lambda) = 0$.

– **Strong Knowledge-Soundness**[8]**.** For all $\lambda \in \mathbb{N}$ and for all (non-uniform) efficient adversaries $\mathcal{A}_{\mathrm{dv}}$ there exists a (non-uniform) efficient extractor $\mathcal{E}_{\mathrm{dv}}$ such that

$$\Pr\left[\begin{array}{c} (\mathsf{ek},\mathsf{vk}) \leftarrow \mathsf{dvKeyGen}(1^{\lambda}) \\ (\mathsf{stmt},\pi) \leftarrow \mathcal{A}_{\mathrm{dv}}^{\mathcal{O}_{\mathrm{dv}}(\mathsf{vk},\cdot)}(z,\mathsf{ek}) \\ \mathsf{w} \leftarrow \mathcal{E}_{\mathrm{dv}}^{\mathcal{O}_{\mathrm{dv}}(\mathsf{vk},\cdot)}(z,\mathsf{ek}) \end{array} : \begin{array}{c} \mathcal{R}(\mathsf{stmt},\mathsf{w}) \neq 1 \wedge \\ \mathcal{V}_{\mathsf{SNARK}}(\mathsf{vk},\mathsf{stmt},\pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O}_{\mathrm{dv}}(\mathsf{vk},\cdot) := \mathcal{V}_{\mathsf{SNARK}}(\mathsf{vk},\cdot)$

– **Succinctness.** For a polynomial p, the running time of $\mathcal{V}_{\mathsf{SNARK}}$ is $o(|\mathcal{C}_{\ell}|) \cdot p(\lambda, |\mathsf{stmt}|)$ and the proof size is $O(\lambda)$.

3 Key-Less Zero-Knowledge

Here we introduce a variant notion of zero-knowledge for dvSNARKs and that will be sufficient in our constructions (section 5.3). We call it *key-less* zero-knowledge and it states that a proof leaks nothing to any adversary without the verification key. This is less stringent than the standard zero-knowledge requirement where we require a proof to leak nothing even to an adversary *holding a verification-key.* We show this weaker notion is sufficient to obtain (full) zero-knowledge in our model as formalized in Definition 5. Below (remark 1) we also argue how it may allow for simpler and more efficient designated-verifier SNARKs to be plugged into our construction.

Definition 2 ((Unbounded) Key-Less Zero-knowledge). *We say dvSNARK* $\Pi_{\mathsf{SNARK}} = (\mathsf{dvKeyGen}, \mathcal{P}_{\mathsf{SNARK}}, \mathcal{V}_{\mathsf{SNARK}})$ *is key-less zero-knowledge if there exists a stateful efficient simulator S such that for all $\lambda \in \mathbb{N}$ and all PPT adversary $\mathcal{A}$, we have that* $|\Pr[\mathsf{klZK}_{\mathcal{A}}^{\mathsf{hon}}(1^{\lambda}) = 1] - \Pr[\mathsf{klZK}_{\mathcal{A}}^{\mathsf{sim}}(1^{\lambda}) = 1]| \leq \mathsf{negl}(\lambda)$*, where* $\mathsf{klZK}_{\mathcal{A}}^{\mathsf{hon}}(1^{\lambda})$ *and* $\mathsf{klZK}_{\mathcal{A}}^{\mathsf{sim}}(1^{\lambda})$ *are defined in fig. 2.*

```
$\underline{\mathsf{klZK}_{\mathcal{A}}^{\mathsf{world}\in\{\mathsf{hon},\mathsf{sim}\}}(1^{\lambda})}$
  $(\mathsf{ek},\mathsf{vk}) \leftarrow \mathsf{dvKeyGen}(1^{\lambda})$
  $\mathsf{guess} \leftarrow \mathcal{A}^{\mathcal{O}_{\text{kl-zk}}^{\mathsf{world}}}(z,\mathsf{ek})$
  Output 1 iff $\mathsf{guess} = \mathsf{world}$
```

[8] The definition is an adaptation of that in [21] and corresponds to a "strong" version of knowledge-soundness for dvSNARKs where we allow the adversary to perform an unbounded number of queries to a verification oracle.

```
$\mathcal{O}^{\mathsf{hon}}_{\text{kl-zk}}(\mathsf{inp})$
    Parse $\mathsf{inp}$ as $(\mathsf{stmt}, \mathsf{w})$
    if $(\mathsf{stmt}, \mathsf{w}) \notin \mathcal{R}$ then return $\perp$
    return $\mathcal{P}_{\mathsf{SNARK}}(\mathsf{ek}, \mathsf{stmt}, \mathsf{w})$

$\mathcal{O}^{\mathsf{sim}}_{\text{kl-zk}}(\mathsf{inp})$
    Parse $\mathsf{inp}$ as $(\mathsf{stmt}, \mathsf{w})$
    if $(\mathsf{stmt}, \mathsf{w}) \notin \mathcal{R}$ then return $\perp$
    return $S(\mathsf{ek}, \mathsf{stmt})$
```

Fig. 2: Key-Less Zero-Knowledge Experiment

Remark 1. The notion above may allow for *simpler* and *more efficient* designated-verifier SNARKs to be plugged in our construction. For example, consider the compiler in [6]. It allows to transform a honest-verifier zero-knowledge (HVZK) linear PCP (a proof system where the verifier has oracle access to a linear function of their queries) into a zero-knowledge dvSNARK. They also show how any linear PCP can be transformed into a HVZK one in a general way. This transformation however occurs at the cost of increasing the number of queries, which concretely implies an increase in proof-size in the final SNARK. We do not need to incur this overhead as we do not require a "fully zero-knowledge" designated-verifier SNARK.

Existence of key-less zero-knowledge dv-SNARKs. We show in Theorem 1 that the dvSNARKs obtained through the compiler in [6] satisfy our weaker requirement even when compiling information-theoretic objects that are not zero-knowledge to start with. The compiler works by letting the evaluation key encrypt queries to an information-theoretic proof system with algebraic properties. A prover holding the witness can use these ciphertexts in the verification key to homomorphically compute an answer for the verifier. The latter decrypts them and performs a test consisting of a few polynomial evaluations.

In the following we use a modified compiler where the prover first rerandomizes the ciphertexts. We can always use rerandomization in this compiler since we assume linearly-homomorphic encryption We note that [6] mentions how rerandomization can be used to achieve zero-knowledge. However, it does not show the weaker type of property we are interested in.

To see the difference between standard zero-knowledge for dv-SNARKs and key-less zero-knowledge we observe the following: in the construction we use below an adversary with a decryption key would be able to learn information about the witness if the underlying NILP (Non-Interactive Linear Proof, a close relative of a linear PCP defined in [34]) is *not* HVZK. On the other hand, we can obtain key-less zero-knowledge for any NILP

Theorem 1. *For any NILP the compiler in [6] produces a dvSNARK with key-less zero-knowledge.*

Proof. The original compiler in [6] has the honest prover homomorphically compute a linear function of a vector of ciphertexts. We let the prover first rerandomize the encryption by adding a an encryption of zero generated with fresh randomness and then compute the same linear function. Simulation follows directly by semantic security of the ciphertexts.

The following theorem shows that we can obtain key-less ZK dvSNARKs from pvSNARKs that are not zero-knowledge. We can obtain this by using the folklore transformation that encrypts a proof with the designated-verifier key.

Theorem 2. *If there exist publicly-verifiable SNARKs (not necessarily zero-knowledge) and public-key encryption then there exist key-less zero-knowledge designated verifier SNARKs.*

Proof. (Sketch) We construct the dvSNARK from a pvSNARK and a PKE as follows: the setup algorithm $\mathsf{dvKeyGen}$ first runs the key generation of PKE and obtains $(\mathsf{pk}, \mathsf{sk})$, and then returns $(\mathsf{ek} := \mathsf{pk}, \mathsf{vk} := \mathsf{sk})$. The prover $\mathsf{dvProve}$ first runs the pvSNARK prover to obtain a proof π. It next computes a ciphertext ct being an encryption of π under pk, and returns ct as the final proof. The verification algorithm first decrypt ct and then runs the pvSNARK verification on the plaintext. Completeness and soundness of the construction follow straightforwardly from the equivalent properties of pvSNARK and correctness of the underlying PKE scheme. Key-less ZK property states that the proof ct should not reveal any information about the prover's witness without the verification key. This follows from the semantic security of PKE.

4 Definition of SPuC

In this section we define our primitive SPuC-s.

Definition 3. *Let $P = [\mathsf{N}]$ be a set of parties. A SPuC Π with a (d, N)-threshold access structure and relation family $(\mathsf{RSet}_\lambda)_{\lambda\in\mathbb{N}}$ with completeness error $c = c(\lambda)$ and soundness error $\epsilon = \epsilon(\lambda)$ is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{PartCert}, \mathsf{PartCertVfy}, \mathsf{Vfy})$ *such that*

– $\mathsf{Setup}(1^\lambda, \mathcal{R}, \mathsf{d}, \mathsf{N}) \rightarrow (\mathsf{pp}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}\})$: On input the description of $\mathcal{R}$ and threshold parameters d, N, the setup algorithm outputs public parameters pp and a set of verification state shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}$.
– $\mathsf{Prv}(\mathsf{pp}, \mathsf{stmt}, \mathsf{w}) \rightarrow \pi_0$: On input pp, a statement stmt and a witness w, the prover algorithm outputs a proof π_0.
– $\mathsf{PartCert}(\mathsf{sk}_i, \mathsf{stmt}, \pi_0) \rightarrow \pi^{(i)}$: On input a verification state share sk_i, a statement stmt and a proof π_0, the partial public prover algorithm outputs a partial proof $\pi^{(i)}$ related to the partial certifier i.
– $\mathsf{PartCertVfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0, \pi^{(i)}) \rightarrow \{0, 1\}$: On input pp, a statement stmt, a proof π_0 and a partial proof $\pi^{(i)}$, the partial verifier outputs a bit $b \in \{0, 1\}$.
– $\mathsf{Vfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0, B) \rightarrow \{0, 1\}$: On input pp, a statement stmt, a proof π_0 and a set $B = \{\pi^{(i)}\}_{i\in I_S}$ for some $S \subseteq [\mathsf{N}]$ with index set I_S, the verifier algorithm outputs a bit $b \in \{0, 1\}$.

Remark 2. Although we do not make it explicit in the syntax, the public parameters can be split in two: prover-related (used in Prv) and verifier-related (used verification algorithms) parameters. The former of size potentially growing with $|\mathcal{R}|$, while the latter of independent size and concretely much smaller.

We require the following properties.

Correctness : For all $\lambda \in \mathbb{N}$, $\mathcal{R} \in \mathsf{RSet}_\lambda$, $(\mathsf{stmt}, \mathsf{w}) \in \mathcal{R}$, any set S with cardinality no smaller than d, we have that the following probability is at least $1 - c(\lambda)$

$$\Pr\left[\begin{array}{r} (\mathsf{pp}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}\}) \leftarrow \mathsf{Setup}(\mathcal{R}, \mathsf{d}, \mathsf{N}) \\ \pi_0 \leftarrow \mathsf{Prv}(\mathsf{pp}, \mathsf{stmt}, \mathsf{w}) \\ \pi^{(i)} \leftarrow \mathsf{PartCert}(\mathsf{sk}_i, \mathsf{stmt}, \pi_0) \end{array} : \mathsf{Vfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0, \{\pi^{(i)}\}_{i \in I_S}) = 1\right]$$

Moreover, for any statement stmt^*, proof π_0^* and for any set of partial proofs $B = \{\pi^{*(i)}\}_{i \in I_S}$ such that $\mathsf{Vfy}(\mathsf{pp}, \mathsf{stmt}^*, \pi_0^*, B) = 1$, it should hold for all $i \in I_S$,

$$\Pr[\mathsf{PartCertVfy}(\mathsf{pp}, \mathsf{stmt}^*, \pi_0^*, \pi^{*(i)}) = 1] \geq 1 - c(\lambda)$$

where $(\mathsf{pp}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}\}) \leftarrow \mathsf{Setup}(\mathcal{R}, \mathsf{d}, \mathsf{N})$.

Succinctness. The running time of Verify is $O_\lambda(\mathsf{d}(|\mathsf{stmt}| + \log(|\mathsf{w}|)))$ and the size of each proof and certificate is $O_\lambda(\log(|\mathsf{w}|))$.

Robustness. We require that for all $\lambda \in \mathbb{N}$, $\mathcal{R} \in \mathsf{RSet}_\lambda$, it holds that for any PPT adversary $\mathcal{A}$, the following experiment called $\mathsf{Expt}_{\mathcal{A},\mathsf{robust}}(1^\lambda)$ outputs 1 with negligible probability.

1. The challenger runs $(\mathsf{pp}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}\}) \leftarrow \mathsf{Setup}(\mathcal{R}, \mathsf{d}, \mathsf{N})$ and then sends $(\mathsf{pp}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}\})$ to $\mathcal{A}$.
2. $\mathcal{A}$ outputs a statement stmt^*, a proof π_0^* and a partial proof $\pi^{*(i)}$.
3. The challenger returns 1 if $\mathsf{PartCertVfy}(\mathsf{pp}, \mathsf{stmt}^*, \pi_0^*, \pi^{*(i)}) = 1$ and $\pi^{*(i)} \neq \mathsf{PartCert}(\mathsf{sk}_i, \mathsf{stmt}^*, \pi_0^*)$.

Knowledge Soundness We require that if an adversary is able to convince the verifier, then we can extract a valid witness from it. Intuition about the experiment: the adversary chooses a corruption set and gets the secret keys for that set. It is then given oracle access to partial proofs from all the other parties.

Definition 4 (Knowledge Soundness). *For all* $\lambda \in \mathbb{N}$*,* $\mathcal{R} \in \mathsf{RSet}_\lambda$ *and for all (non-uniform) efficient stateful adversaries* $\mathcal{A}$ *there exists a (non-uniform) efficient extractor* $\mathcal{E}$ *such that* $\Pr[\mathsf{KSND}_{\mathcal{A},\mathcal{E}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$

```
$\mathsf{KSND}_{\mathcal{A},\mathcal{E}}(1^\lambda)$
    $(\mathsf{pp}, \mathbf{sk} = (\mathsf{sk}_1, \dots, \mathsf{sk}_\mathsf{N})) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, \mathsf{d}, \mathsf{N})$
    $C \leftarrow \mathcal{A}(\mathsf{pp})$ where $|C| = \mathsf{d} - 1$
    $(h, \mathsf{stmt}, \pi_0, \pi_{i_1}, \dots, \pi_{i_{\mathsf{d}-1}}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{prf}}}(\mathsf{pp}, (\mathsf{sk}_j)_{j \in C})$ where $h \in [\mathsf{N}] \setminus C$
    $\pi_h \leftarrow \mathsf{PartCert}(\mathsf{sk}_h, \mathsf{stmt}, \pi_0)$
    $\mathsf{w} \leftarrow \mathcal{E}^{\mathcal{O}_{\mathrm{prf}}}(\mathsf{pp})$
    Output 1 iff $\mathcal{R}(\mathsf{stmt}, \mathsf{w}) \neq 1 \wedge \mathsf{Verify}(\mathsf{pp}, \pi_0, (\pi_h, \pi_{i_1}, \dots, \pi_{i_{\mathsf{d}-1}})) = 1$
```

The oracle $\mathcal{O}_{\mathrm{prf}}$ above works as follows: given a pair (stmt', π_0') the adversary is given all the responses $\mathsf{PartCert}(\mathsf{sk}_i, \mathsf{stmt}', \pi_0')$ for $i \in [\mathsf{N}]$.
NB: above, the extractor does not need to take as input π_h since it can always obtain it from the (deterministic) proof oracle it has access to by emulating the adversary's behavior. This approach to modeling the extractor has the advantage of not requiring an explicit trapdoor (we remark that constructions of this type are possible [6]), thus allowing for a somewhat stronger notion. We follow a similar line of modeling when defining strong knowledge-soundness for dvSNARKs (definition 1).

Zero-Knowledge. In the zero-knowledge experiment we let the adversary to corrupt a certain subset of parties and then access to two types of oracles:

– one in which it supplies a statement stmt (not necessarily in the language) and some π_0 and gets the partial certificates from all the secret key holders;
– one analog to the oracle for standard zero-knowledge where, given a pair statement–witness satisfying the relation, it receives a proof together with certificates for it.

Definition 5 (Zero-Knowledge). *We say SPuC with a (d, N)-threshold access structure is zero-knowledge if there exists a stateful efficient simulator tuple $S = (S_1, S_{prt}, S_{prf})$ such that for all $\lambda \in \mathbb{N}$, all $\mathcal{R} \in \mathsf{RSet}_\lambda$ and all PPT adversary $\mathcal{A}$, we have that*

$$|\Pr[\mathsf{ZK}^{\mathsf{hon}}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

where the experiments are defined in Figure 3.

5 Construction of SPuC

In this section we describe constructions of (non-proactive) SPuC and discuss its instantiations. The goal of subsection 5.1 is to serve as a warm-up to some of the challenges of constructing SPuC-s and informally describes a limited construction. We provide preliminaries for our general SPuC construction—universal

```
$\mathsf{ZK}_{\mathcal{A}}^{\mathsf{world}\in\{\mathsf{hon},\mathsf{sim}\}}(1^\lambda)$
  if $\mathsf{world} = \mathsf{hon}$ then
    $(\mathsf{pp}, \mathbf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, \mathsf{d}, \mathsf{N})$
  else
    $(\mathsf{pp}, \mathbf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})) \leftarrow S_1(\mathcal{R}, \mathsf{d}, \mathsf{N})$
  $C \leftarrow \mathcal{A}(\mathsf{pp})$ where $|C| = \mathsf{d} - 1$
  $\mathsf{guess} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{zk}}^{\mathsf{world}}}(\mathsf{pp}, (\mathsf{sk}_j)_{j\in C})$
  Output 1 iff $\mathsf{guess} = \mathsf{world}$
```

```
$\mathcal{O}_{\mathrm{zk}}^{\mathsf{hon}}(\mathsf{tag}, \mathsf{inp})$
  if $\mathsf{tag} = \mathsf{part\text{-}proofs}$ then
    Parse $\mathsf{inp}$ as $(\mathsf{stmt}, \pi_0)$
    $(\pi_i)_{i\in[\mathsf{N}]} \leftarrow \big(\mathsf{PartCert}(\mathsf{sk}_i, \mathsf{stmt}, \pi_0)\big)_{i\in[\mathsf{N}]}$
    return $(\pi_1, \ldots, \pi_\mathsf{N})$
  if $\mathsf{tag} = \mathsf{valid\text{-}x}$ then
    Parse $\mathsf{inp}$ as $(\mathsf{stmt}, \mathsf{w})$
    if $(\mathsf{stmt}, \mathsf{w}) \notin \mathcal{R}$ then return $\perp$
    return $\mathsf{Prv}(\mathsf{pp}, \mathsf{stmt}, \mathsf{w})$
```

```
$\mathcal{O}_{\mathrm{zk}}^{\mathsf{sim}}(\mathsf{tag}, \mathsf{inp})$
  if $\mathsf{tag} = \mathsf{part\text{-}proofs}$ then
    Parse $\mathsf{inp}$ as $(\mathsf{stmt}, \pi_0)$
    $(\pi_i)_{i\in[\mathsf{N}]} \leftarrow \big(S_{\mathrm{prt}}(\mathbf{sk}, \mathsf{stmt}, \pi_0)\big)_{i\in[\mathsf{N}]}$
    return $(\pi_1, \ldots, \pi_\mathsf{N})$
  if $\mathsf{tag} = \mathsf{valid\text{-}x}$ then
    Parse $\mathsf{inp}$ as $(\mathsf{stmt}, \mathsf{w})$
    if $(\mathsf{stmt}, \mathsf{w}) \notin \mathcal{R}$ then return $\perp$
    return $S_{\mathrm{prf}}(\mathbf{sk}, \mathsf{stmt})$
```

Fig. 3: ZK experiment. Oracles take as input $\mathsf{tag} \in \{\mathsf{part\text{-}proofs}, \mathsf{valid\text{-}x}\}$ and some stmt whose structure depends on the tag.

thresholdizers, UT, and designated-verifier SNARKs—in subsection 5.2. We then proceed to describe two instantiations of UT, both based on lattices. In subsection 5.5.1 we present a general construction (no limitations on threshold and number of parties) from Threshold FHE and context-hiding homomorphic signatures (HS). We present a simpler, more efficient construction for the two-party case based on homomorphic-secret sharing (HSS) in subsection 5.5.2.

5.1 Warm-up: a Straw-Man Construction

The following construction—based only on the existence of (zero-knowledge) designated-verifier SNARKs—exemplifies some of the properties we desire in a succinct publicly-certifiable scheme. Although arguably simpler than our other constructions we find it to have stronger limitations, discussed below. Thus we keep its presentation informal.

Assuming the existence of a designated-verifier SNARK scheme (see next section), we can construct a SPuC with N certifiers and threshold d as follows.

- At setup time we generate N different setups $(\mathsf{ek}_i, \mathsf{vk}_i) \leftarrow \mathsf{dvKeyGen}(1^\lambda, \mathcal{R})$, publish the N evaluation keys and a secret verification key vk_i to each of the committee members.
- The algorithm Prv would then produce N designated-verifier proofs π_i^{dv}, each with a different evaluation key ek_i for $i \in [\mathsf{N}]$.

– Each certifier i in the committee (algorithm $\mathsf{PartCert}$) would return a bit stating acceptance or rejection of the respective π_i^{dv} using vk_i and signed with a key of the respective committee member.
– Given a set B of acceptance/rejection bits of size at least d, a verifier would then accept if all the bits in B are 1 and otherwise reject.

For simplicity we have not presented algorithm $\mathsf{PartCertVfy}$ which can be achieved with techniques similar to ours. The construction just described satisfies knowledge soundness and zero-knowledge. its main limitations are a high concrete and asymptotic efficiency and that it is not immediate how to extend it efficiently to a proactively secure construction. For efficiency, notice that we require N designated-verifier setups, which is very expensive (especially if we want to replace the setup stage with an MPC execution). It is also expensive in practice to require that a prover would run N times the proving algorithm. The construction does not technically satisfy succinctness for the same reason: the output of Prv depends on the number of shares. Even if this were acceptable asymptotically (e.g., considering N a constant) this incurs high concrete costs. In addition and in contrast to our constructions, it forces the runner of the algorithm Prv to store N evaluation keys (each of size at least linear in the size of the relation $\mathcal{R}$). This dependency on N is less problematic if this parameter is small. Extending this construction to the proactive case would seem to require regenerating the verification keys (the adversary could learn d of them through different epochs and so they cannot remain the same). It is unclear how to perform this new setup without an interactive MPC or a trusted authority.

We now describe the building blocks for our general construction.

5.2 Building Block Primitive: Universal thresholdizers (UT).

Universal thresholdizers (UTs) are a primitive that can be used to thresholdize a system. A UT scheme with a (d, N)-threshold access structure consists of four algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}, \mathsf{Combine})$. The setup algorithm Setup takes in a secret value x and divides it into a set of shares $\mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}$, which are given to N users. Each user, on input a circuit C, calls Eval and uses their shares s_i to compute an evaluation share y_i of $C(x)$. The verification algorithm Verify can be used to check whether y_i was computed correctly. Finally, for a set $B = \{\mathsf{y}_i\}$ for which $|B| \geq \mathsf{d}$, the algorithm $\mathsf{Combine}$ can be used to combine these evaluation shares and produce $\mathsf{y} = C(x)$.

For a UT scheme to be secure, it should hold that the shares $\mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}$, together with the evaluation shares y_i can be simulated only given access to the circuit C and its output on the secret value x (i.e., $C(x)$). In addition, the robustness property states that no PPT adversary should be able to produce an incorrectly computed evaluation share y_i for a circuit C if the verification algorithm Verify accepts it. For more formal details and constructions on UT see appendix B.

```
$\mathsf{Setup}(1^\lambda, \mathcal{R}, \mathsf{d}, \mathsf{N})$
    $\mathsf{dvKeyGen}(1^\lambda, \mathcal{R}) \to (\mathsf{ek}, \mathsf{vk})$
    $(\mathsf{pp}_{\mathsf{UT}}, \mathsf{sk}_{\mathsf{UT},1}, \ldots, \mathsf{sk}_{\mathsf{UT},\mathsf{N}}) \leftarrow \mathsf{UT.Setup}(\mathsf{d}, \mathsf{N}, \mathsf{vk})$
    return $(\mathsf{pp} = (\mathsf{ek}, \mathsf{pp}_{\mathsf{UT}}), \{\mathsf{sk}_i = \mathsf{sk}_{\mathsf{UT},i}\}_{i \in [\mathsf{N}]})$
```

```
$\mathsf{Prv}(\mathsf{pp} = (\mathsf{ek}, \mathsf{pp}_{\mathsf{UT}}), \mathsf{stmt}, \mathsf{w})$
    return $\pi_0 \leftarrow \mathsf{dvProve}(\mathsf{ek}, \mathsf{stmt}, \mathsf{w})$
```

```
$\mathsf{Vfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0, (\pi^{(i)})_{i \in I_S})$
    return $\mathsf{UT.Combine}(\mathsf{pp}_{\mathsf{UT}}, (\pi^{(i)})_{i \in I_S})$
```

```
$\mathsf{PartCert}(\mathsf{sk}_i, \mathsf{stmt}, \pi_0)$
    $\pi^{(i)} \leftarrow \mathsf{UT.PartEval}(\mathsf{sk}_{\mathsf{UT},i}, C_{\mathsf{stmt},\pi_0})$
    s.t. $C_{\mathsf{stmt},\pi_0}(\cdot) := \mathsf{dvVerify}(\cdot, \mathsf{stmt}, \pi_0)$
```

```
$\mathsf{PartCertVfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0, \pi^{(i)})$
    return $\mathsf{UT.Verify}(\mathsf{pp}_{\mathsf{UT}}, C_{\mathsf{stmt},\pi_0}, \pi^{(i)})$
    s.t. $C_{\mathsf{stmt},\pi_0}(\cdot) := \mathsf{dvVerify}(\cdot, \mathsf{stmt}, \pi_0)$
```

Fig. 4: Construction of SPuC from UT

Theorem 3 (Implicit in [9]). *If there exists leveled Threshold FHE and compact context-hiding homomorphic signatures (HS) then there exists UT. It is possible to construct leveled Threshold FHE from LWE.*

5.3 A General Construction for SPuC

Our construction is in Figure 4.

Theorem 4. *(informal) If there exists UT and zero-knowledge dvSNARKs then there exists a secure SPuC.*

5.4 Proof of Security

Completeness and robustness follow straightforwardly from the equivalent properties of UT [9]. We prove knowledge soundness and zero-knowledge.

Lemma 1 (Knowledge Soundness). *The construction in Figure 4 is knowledge sound (definition 4) if UT is Universal Thresholdizer and DV is a designated-verifier SNARK with strong knowledge-soundness (definition 1).*

Proof. Consider an adversary $\bar{\mathcal{A}}$ in the knowledge soundness experiment of Definition 4 for some $\lambda, \mathsf{N} \in \mathbb{N}$. Let us construct an adversary $\mathcal{A}_{\text{dv}}$ for the strong knowledge-soundness experiment as in Figure 5. We construct an extractor $\bar{\mathcal{E}}$ that internally runs the knowledge soundness extractor $\mathcal{E}_{\text{dv}}$, corresponding to $\mathcal{A}_{\text{dv}}$. We claim that the extractor outputs a witness with high probability if $\bar{\mathcal{A}}$ produces a valid proof with high probability. First observe that (x, π_0), output of $\mathcal{A}_{\text{dv}}$, must verify successfully with probability negligibly close to that of $\bar{\mathcal{A}}$. This follows from the definition of $S = (S_1^{\text{UT}}, S_2^{\text{UT}})$, security of UT as well as its verification and evaluation correctness: the output of $\mathcal{O}'$ in $\mathcal{A}_{\text{dv}}$ must be

```
$\mathcal{A}_{\mathrm{dv}}^{\mathcal{O}_{\mathrm{dv}}}(\mathsf{ek})$
  Let $S = (S_1^{\mathrm{UT}}, S_2^{\mathrm{UT}})$ be the simulator from the UT security;
  $(\mathsf{pp}, s_1, \ldots, s_{\mathsf{N}}, st) \leftarrow S_1^{\mathrm{UT}}(1^\lambda)$;  $C \leftarrow \bar{\mathcal{A}}(\mathsf{pp})$
  Define oracle $\mathcal{O}'(\mathsf{stmt}', \pi_0')$ as :
    $b \leftarrow \mathcal{O}_{\mathrm{dv}}(\mathsf{stmt}', \pi_0')$
    $(\pi_i)_{i \in [\mathsf{N}]} \leftarrow S_2^{\mathrm{UT}}(\mathsf{pp}, C_{\mathsf{stmt}', \pi_0'}, b, st)$
      where $C_{\mathsf{stmt}', \pi_0'}$ is the verification circuit (as in construction)
    return $(\pi_i)_{i \in [\mathsf{N}]}$
  $(h, \mathsf{stmt}, \pi_0, \pi_{i_1}, \ldots, \pi_{i_{\mathsf{d}-1}}) \leftarrow \bar{\mathcal{A}}^{\mathcal{O}_{\mathrm{prf}}}(\mathsf{pp}, (s_i)_{i \in C})$
  return $(\mathsf{stmt}, \pi_0)$

$\bar{\mathcal{E}}^{\mathcal{O}_{\mathrm{prf}}}(\mathsf{pp})$
  Define oracle $\mathcal{O}''(\mathsf{stmt}'', \pi_0'')$ as :
    $(\pi_i)_{i \in [\mathsf{N}]} \leftarrow \mathcal{O}_{\mathrm{prf}}(\mathsf{stmt}'', \pi_0'')$
    Find set of $\mathsf{d}$ proofs $\boldsymbol{\pi}^*$ s.t. $\mathsf{SPuC.PartCertVfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0'', \pi_j^*) = 1\ \forall j \in [\mathsf{d}]$
    If $\exists \boldsymbol{\pi}^*$ output $\mathsf{SPuC.Vfy}(\mathsf{pp}, \mathsf{stmt}, \pi_0'', \boldsymbol{\pi}^*)$;  o.w. output 0
  $\mathsf{w} \leftarrow \mathcal{E}_{\mathrm{dv}}^{\mathcal{O}''}(\mathsf{ek})$
  return $\mathsf{w}$
```

Fig. 5: Construction of $\mathcal{A}_{\mathrm{dv}}$ in the proof of lemma 1

computationally indistinguishable from $\mathsf{dvVerify}(\mathsf{vk}, \cdot)$ (the oracle $\mathcal{O}_{\mathrm{dv}}$ in Strong Knowledge-Soundness definition) otherwise we would be able to distinguish between the simulated π_i-s and the honestly computed ones in UT security. By definition of $\mathcal{A}_{\mathrm{dv}}$ the output of $\bar{\mathcal{E}}$ must be a valid witness with probability close to that of $\mathcal{E}_{\mathrm{dv}}$. To show why, we observe that the oracle $\mathcal{O}''$ in the extractor $\bar{\mathcal{E}}$ must have, by construction, an output indistinguishable from that of $\mathsf{dvVerify}(\mathsf{vk}, \cdot)$; we can conclude this by invoking verification and evaluation correctness. □

Lemma 2 (Zero-Knowledge). *The construction in Figure 4 is zero-knowledge (Definition 5) if UT is Universal Thresholdizer and DV is a designated-verifier SNARK with key-less zero-knowledge (Definition 2).*

Proof. Our goal is to build $S = (S_1, S_{\mathrm{prt}}, S_{\mathrm{prf}})$ where S_1 is the simulator for the setup. We shall do that by invoking the security definition of UT [9] and the definition of key-less zero-knowledge (Definition 2). From these theorems it

[9] For the definition of UT security, we refer to the sUT security in fig. 7. Note that UT security is a special case where there is no trapdoor evaluation oracle.

follows the existence of simulators respectively $S^{\mathrm{UT}} = (S_1^{\mathrm{UT}}, S_2^{\mathrm{UT}})$ and S^{klzk}. We then define S_1 so that: it first runs $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{dvKeyGen}(\mathcal{R})$; then $(\mathsf{pp}_{\mathsf{UT}}, \mathbf{sk}) \leftarrow S_1^{\mathrm{UT}}(1^\lambda, \mathsf{N})$; then it outputs a public key $(\mathsf{ek}, \mathsf{pp}_{\mathsf{UT}})$ and a simulation trapdoor $(\mathbf{sk}, \mathsf{vk})$. We shall then define S as $S := (S_1, S_{\mathrm{prt}} = S_2^{\mathrm{UT}}, S_{\mathrm{prf}} = S^{\mathrm{klzk}})$.

Claim 1: The output of S_1 is indistinguishable from that of the honest setup. This follows directly from the definition of S_1^{UT} and from UT security.

Hybrid Experiments. Let $q(\lambda) = \mathsf{poly}(\lambda)$ be an upper bound on the number of oracle queries of the adversary in the experiment in Definition 5. For each $i \in \{0, 1, \ldots, q(\lambda)\}$ we define a hybrid zero-knowledge experiment $\mathcal{H}_i$ as in figure. For all oracle queries $j \in [q(\lambda)]$ the oracle $\mathcal{O}^i_{\mathrm{zk}}$ acts as follows: for query $j \leq i$ the adversary receives honest generated queries (from $\mathcal{O}^{\mathsf{hon}}_{\mathrm{zk}}$); for query $j > i$ receives simulated queries ($\mathcal{O}^{\mathsf{sim}}_{\mathrm{zk}}$). Notice that hybrid $\mathcal{H}_0$ corresponds to $\mathsf{ZK}^{\mathsf{hon}}$ and hybrid $\mathcal{H}_{q(\lambda)}$ to $\mathsf{ZK}^{\mathsf{sim}}$. It is now sufficient to prove the following claim.

Claim 2: for all $i \in [q(\lambda)]$ $\mathcal{H}_{i-1} \approx \mathcal{H}_i$. Notice that the oracles in Figure 3 are such that for any $\mathsf{tag} \in \{\mathsf{part\text{-}proofs}, \mathsf{valid\text{-}x}\}$ $\mathcal{O}^{\mathsf{hon}}_{\mathrm{zk}}(\mathsf{tag}, \mathsf{inp}) \approx \mathcal{O}^{\mathsf{sim}}_{\mathrm{zk}}(\mathsf{tag}, \mathsf{inp})$. This is because of the security of the UT and the (key-less) zero-knowledge property of the dvSNARK. If two consecutive hybrids were distinguishable then it would be possible to distinguish either of the two oracles with non-negligible probability since the $i-1$ queries can be efficiently implemented.

$\mathcal{H}^i_{\mathcal{A}}(1^\lambda, \mathsf{N})$

$(\mathsf{pp}_{\mathsf{UT}}, \mathbf{sk} = (sk_1, \ldots, sk_{\mathsf{N}})) \leftarrow S_1^{\mathrm{UT}}(1^\lambda, \mathcal{R}, \mathsf{N})$; $C \leftarrow \mathcal{A}(\mathsf{pp})$ where $|C| = \mathsf{d} - 1$

Output $\mathsf{guess} \leftarrow \mathcal{A}^{\mathcal{O}^i_{\mathrm{zk}}}(\mathsf{pp}_{\mathsf{UT}}, (s_j)_{j \in C})$ where $h \in [\mathsf{N}] \setminus C$

□

5.5 Construction of UT

5.5.1 A General Construction of UT from TFHE and HS ([9])

As informally described in [9], we can construct UT from TFHE and context-hiding HS. TFHE can itself be built from FHE with moderate "noise bound" (roughly, a measure of the noise at the decryption stage), which we can obtain for our purposes from LWE [9]. In appendix B.2 we formally build for the first time UT from homomorphic signatures ([9] only contains formal description and proofs for a pvNIZK-based construction).

Intuition on construction. We exploit Threshold FHE, where one can encrypt a message x ($\mathsf{TFHE.Enc}$), publicly obtain a ciphertext of an evaluation $C(x)$ for a circuit C, members of a committee can provide partial decryptions of a ciphertext through (a share of) a secret key, which can then be publicly combined to obtain a plaintext. When we run $\mathsf{UT.Setup}(x)$ we encrypt the secret x through the TFHE and provide a share of the TFHE secret key to each of the committee members. The evaluation and combination algorithm of UT invoke respectively the partial decryption and combination algorithm of TFHE. To provide robustness we use homomorphic signatures: we let each committee member sign the output of the partial decryption. They can carry this out homomorphically (using $\mathsf{HS.Eval}$), as they are given a signature of their secret key share at setup time (through $\mathsf{HS.Sign}$).

Corollary 1. *(informal) If there exists leveled Threshold FHE for* NC^1*, compact context-hiding Homomorphic Signatures and zero-knowledge dvSNARKs then there exists a secure SPuC.*

Efficiency We can instantiate our construction with dvSNARKs obtained through Square-Span Programs [23] compiled with the results in [6] and homomorphic signatures from [32]. The output of Prove consists of a constant number of ciphertexts each encrypting a field element. Its total size would then be $O_\lambda(1)$.

The size of each certificate is $O_\lambda(\log(|\mathsf{stmt}|))$ which we can derive as follows. First observe that the signatures in [32] after evaluation remain of a size lower than some bound on the depth of the homomorphic computation. Looking inside the TFHE-based UT construction from [9] (formalized in appendix B) and the compiler in [6], we see that the homomorphic computation consists of a partial TFHE decryption on top of a procedure f_{dv}. On input a signed secret of size $O_\lambda(|\mathsf{stmt}|)$ procedure f_{dv} decrypts the aforementioned ciphertexts and performs a zero-test on a low-degree multivariate polynomial with $O(n)$ variables. Hence a bound on the depth of f_{dv} is $O_\lambda(\log(|\mathsf{stmt}|))$. The partial decryption on top of it adds a factor $\mathsf{poly}(\lambda)$.

5.5.2 A Construction of UT from HSS and HS for the (2,2) Setting

In Figure 6 we describe a novel construction for (two-party) UT based on Homomorphic Secret-Sharing [14]. It works similarly to the construction from TFHE. We recall that HS denotes the homomorphic signature scheme and we denote by using $\mathsf{HS.Eval}$ and by $\mathsf{HS.Sign}$ respectively the algorithms for homomorphic evaluation of signatures and for initially signing a message. We can instantiate HSS[10] from LWE through the construction in [16]. While the (2,2)-case for UT is subsumed by the general construction from 5.5.1, our HSS-based construction requires simpler and more efficient primitives (see discussion of efficiency of TFHE vs HSS in [16]). Moreover, although our main focus is quantum-resistant constructions, HSS allows for a wider type of instantiations, for example from DDH as in [14][11] (not known to imply (leveled) FHE).

[10] A (2-party) HSS consists of algorithms: Share to secret share a message, Eval to homomorphically produce a partial evaluation of a function f on the message x given a share, $\mathsf{Combine}$ to publicly recombine the evaluation shares into $f(x)$.

[11] This instantiation is still plausibly weaker than publicly-verifiable NIZKs; the recent breakthrough in [36] requires a *sub-exponential* version of DDH to build pvNIZKs.

```
$\mathsf{UT}_{(2,2)}.\mathsf{Setup}(1^\lambda, \mathsf{d} = 2, \mathsf{N} = 2, x)$
    $(\mathsf{sk_{hs}}, \mathsf{pk_{hs}}) \leftarrow \mathsf{HS.Setup}(1^\lambda)$
    $\mathbf{sk} \leftarrow \mathsf{UT}_{(2,2)}\text{-}\mathsf{AuxSetup}(x, \mathsf{sk_{hs}})$
    return $(\mathsf{pp} := \mathsf{pk_{hs}}, \mathbf{sk})$

$\mathsf{UT}_{(2,2)}\text{-}\mathsf{AuxSetup}(x, \mathsf{sk_{hs}})$
    $(\mathsf{share}_1, \mathsf{share}_2) \leftarrow \mathsf{HSS.Share}(1^\lambda, x)$
    for $i = 1, 2$ :
        $\mathsf{sk}'_i[\mathsf{hs}] \leftarrow \mathsf{HS.Sign}(\mathsf{sk_{hs}}, “i”, \mathsf{share}_i)$
        $\mathsf{sk}'_i := (\mathsf{share}_i, \mathsf{sk}'_i[\mathsf{hs}])$
    return $(\mathsf{sk}'_1, \mathsf{sk}'_2)$

$\mathsf{UT}_{(2,2)}.\mathsf{PartEval}(\mathsf{pp} := \mathsf{pk_{hs}}, \mathsf{sk}_i, i, C)$
    $y_i \leftarrow \mathsf{HSS.Eval}_i(\mathsf{share}_i, C)$
    $\sigma_i \leftarrow \mathsf{HS.Eval}(\mathsf{pk_{hs}}, “i”, C_{\mathrm{HSSEval}}, \mathsf{sk}_i[\mathsf{hs}])$
        where $C_{\mathrm{HSSEval}} := \mathsf{HSS.Eval}_i(\cdot, C)$
    return $(y_i, \sigma_i)$

$\mathsf{UT}_{(2,2)}.\mathsf{Combine}\,(\mathsf{pp} := \mathsf{pk_{hs}}, y_1, y_2)$
    return $\mathsf{HSS.Combine}(y_1, y_2)$.

$\mathsf{UT}_{(2,2)}.\mathsf{VfyEval}\,(\mathsf{pp} := \mathsf{pk_{hs}}, \pi_i = (y_i, \sigma_i), i, C)$
    return $\mathsf{HS.Verify}(\mathsf{pk_{hs}}, “i”, y_i, \sigma_i, C_{\mathrm{HSSEval}})$
        where $C_{\mathrm{HSSEval}} := \mathsf{HSS.Eval}_i(\cdot, C)$
```

Fig. 6: 2-party UT Construction, $\mathsf{UT}_{(2,2)}$, from HSS and HS.

Corollary 2. *(informal) If there exists two-party-HSS for* NC^1*, context-hiding HS and zero-knowledge dvSNARKs then there exists a two-party SPuC.*

Remark 3 (On UT and Robust HSS). We observe that the notion of UT is very close to *robust* homomorphic secret-sharing scheme (see, e.g., Section 2 in [15]). We, however, present it in the language of UT because it allows to use for the same framework as that of our Section 5.5.1 and for continuity with [9].

6 Proactive UT and Proactive SPuC

We define a new primitive pUT, proactive version of UT where the committee members can change constantly. The protocol is divided in epochs with a handover stage at the end of each. During each epoch t, the members of committee ($\mathcal{C}_t$) can carry out oblivious evaluations as in UT and later hand over their shares to the next committee $\mathcal{C}_{t+1}$. We require these steps to be non-interactive and robust (roughly, the resharing phase should be publicly verifiable).

After being nominated (a nomination stage is out of the scope of this paper and we merely posit it) the committee member i for the next epoch holds an ephemeral secret key esk_i. Its share of the secret will be encrypted with a corresponding ephemeral public key epk_i. For this purpose a pUT is coupled with a public-key encryption scheme PK.

Here we present an overview of the model and the construction. Further details can be found in appendix D.

6.1 Proactive UT: Model Description

A pUT extends the syntax of UT with algorithms for resharing, reconstruction and related verification: – $\mathsf{pUT.Reshare}(\mathsf{pp}, \mathsf{sk}_i^t, i, \mathbf{epk}^{t+1}) \rightarrow (y_i^{\text{resh}}, \sigma_i^{\text{resh}})$: using a partial secret key sk_i^t this algorithm performs a (partial) handover of secret i to the committee in epoch $t+1$. – $\mathsf{pUT.VfyReshare}(\mathsf{pp}, \mathbf{epk}^t, (y_i^{\text{resh}}, \sigma_i^{\text{resh}}), i) \rightarrow \{0,1\}$: The algorithm verifies if party i carried out a resharing step correctly. – $\mathsf{pUT.Reconstruct}(\mathsf{pp}, \mathsf{esk}_j^t, (y_i^{\text{resh}})_{i\in[\mathsf{d}]}) \rightarrow \mathsf{sk}_j^{t+1}$: Having d shares (y_i^{resh}), the algorithm reconstruct a secret share sk_j^{t+1} through esk_j.

6.2 Building Block: Special UT (sUT)

We construct pUT from another novel primitive, sUT. If a pUT extends UT with resharing features, a sUT extends it with a special type of oblivious evaluation. Recall that in UT committee members can obliviously compute functions on a secret x, provided as *input* to the UT setup. In sUT, on the other hand, we also allow to compute functions on secrets of the sUT itself (a trapdoor generated at setup time). This very powerful type of evaluation will be useful in pUT to reshare the trapdoor itself. Naturally we need to somehow constrain the type of evaluations allowed to the adversary. In order to do this we allow two types of evaluation queries: one unconstrained (on the secret x) and one (on the sUT trapdoor) with respect to a circuit sampler. For more formal details on the definition and security requirements of sUT see appendix C.1.

Definition 6 (Circuit Sampler). *A circuit sampler D is a PPT that on input a string z returns a circuit $C \leftarrow D(z)$ of size polynomial in $|z|$.*

Definition 7. *(sUT Security) A sUT scheme satisfies security with respect to circuit sampler D if there exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_S, \mathcal{S}_E)$ such that for all λ, for any PPT adversary $\mathcal{A}$, the following experiments $\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right) \approx \mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ (see Figure 7).*

6.2.1 A Construction for sUT.

We extend the construction from [9] to prove evaluations on the sUT trapdoor. The $\mathsf{TrapdEval}$ function works exactly as $\mathsf{PartEval}$ but on a different ciphertext (which encrypts the trapdoor). The construction is using almost the same assumptions as for UT, namely homomorphic signatures and FHE, but we need to also assume circular security of the latter as we should be able to securely encrypt its own decryption key. A construction of sUT is in Figure 8. We give more details of the construction and its security in appendix C.2.

6.3 Construction of pUT

We give a construction based on a homomorphic signature HS, a threshold fully homomorphic encryption scheme TFHE, and PRFs PRF. Our construction is in

$\underline{\mathsf{Expt}^{\mathsf{world}\in\{\mathsf{real},\mathsf{ideal}\}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda,\mathsf{d},\mathsf{N}\right)}$:

1. $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{d}, \mathsf{N})$
2. if $\mathsf{world} = \mathsf{real}$ then $(\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right)$
3. else if $\mathsf{world} = \mathsf{ideal}$ then $(\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathcal{S}_S\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$.
4. $\mathcal{A}$ outputs a corruption set $\mathcal{C}$ of size $\mathsf{d} - 1$
5. The challenger provides the shares $\{\mathsf{s}_i\}_{i\in\mathcal{C}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ can ask for a polynomial number of adaptive queries to the oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{sUT}}$ (defined below).
7. Adversary outputs a guess guess
8. Return 1 iff $\mathsf{guess} = \mathsf{world}$

The oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{sUT}}$ can receive in input either a tuple $(\mathsf{trapdquery}, z)$ or a tuple (xquery, C). The first asks for a query evaluation on the trapdoor; the other for the secret x. For the case $(\mathsf{trapdquery}, z)$, the oracle samples a circuit from sampler D as $C \leftarrow\!\!\$\ D(\mathsf{trapd}, z)$, returns circuit C and partial evaluation $\{\mathsf{y}_i \leftarrow \mathsf{sUT.TrapdEval}\left(\mathsf{pp}, \mathsf{s}_i, C\right)\}_{i\in[\mathsf{N}]}$. For the case (xquery, C) the oracle responds with:

– if $\mathsf{world} = \mathsf{real}$ then return $\{\mathsf{y}_i \leftarrow \mathsf{sUT.Eval}\left(\mathsf{pp}, \mathsf{s}_{i,T}, C\right)\}_{i\in[\mathsf{N}]}$
– if $\mathsf{world} = \mathsf{ideal}$ then return $\{\mathsf{y}_i\}_{i\in[\mathsf{N}]} \leftarrow \mathcal{S}_E\left(\mathsf{trapd}, C, C(x)\right)$

Fig. 7: Security experiment for sUT

fact based on the sUT construction by applying the algorithm TrapEval of pUT on a "resharing" function $\mathcal{F}^{\mathrm{resh}}_{t,\mathbf{epk}^{t+1}}$, tied to our sUT construction (see also Fig. 9) that generates a new secret and then creates, signs and encrypts its shares for the next epoch. Other techniques of our construction for pUT are inspired by the YOSO-style ones in [3] where the committee members of the new epoch can access their share by opening a ciphertext encrypted with an ephemeral public key (of which they only know the decryption key). The main intuition is that a committee member can carry out homomorphic computation on the encrypted secrets and then certify through homomorphic signatures their partial decryption. The result can publicly be combined to obtain the function output. The construction of pUT is in Fig. 9.

Theorem 5. *(Informal) We can construct sUT for a "family of resharing functions" from compact context-hiding homomorphic signatures and leveled TFHE with KDM security [7, 11]. We can construct pUT from the same assumptions.*

6.4 From pUT to pSPuC

Proactive SPuCs extend the SPuC model in the same way as pUT extends UT. A pSPuC includes algorithms (Reshare, VfyReshare, Reconstruct) to allow the committee members to hand over their secrets for certification. Once defined (and constructed) pUTs, a construction for pSPuCs is straightforward: it is the same as the one for SPuCs, but we replace UT with pUT.

Acknowledgements We thank the anonymous reviewers, as well as Jesper Buus Nielsen, Mahak Pancholi and Antonio Faonio for useful discussions around this work. Matteo Campanelli was supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM). Hamidreza Khoshakhlagh was funded by the Concordium Foundation under Concordium Blockchain Research Center, Aarhus.

```
$\mathsf{sUT.Setup}(1^\lambda, \mathsf{d}, \mathsf{N}, x)$
    $(\mathsf{sk_{hs}}, \mathsf{pk_{hs}}) \leftarrow \mathsf{HS.Setup}(1^\lambda)$
    $(\mathsf{sk_{fhe}}, \mathsf{pk_{fhe}}) \leftarrow \mathsf{TFHE.KeyGenAux}(1^\lambda, \mathsf{d}, \mathsf{N})$
    $\mathsf{ct}_x \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk_{fhe}}, x)$
    $\rho \leftarrow\!\!\$\ \mathsf{rnd}$
    $\mathsf{trapd} := (\mathsf{sk_{fhe}}, \mathsf{sk_{hs}}, \rho)$
    $\mathbf{sk} \leftarrow \mathsf{AuxSetup}(\mathsf{trapd})$
    $\mathsf{ct}_{\mathrm{tpd}} \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk_{fhe}}, \mathsf{trapd})$
    return $(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}, \mathsf{ct}_x, \mathsf{ct}_{\mathrm{tpd}}), \mathbf{sk}, \mathsf{trapd})$

$\mathsf{AuxSetup}(s)$
    Parse $s$ as $s = (\mathsf{sk_{fhe}}, \mathsf{sk_{hs}}, \rho)$
    Parse $\rho$ as $\rho = (\rho[\mathsf{SS}], \rho_1[\mathsf{hs}], \ldots, \rho_{\mathsf{N}}[\mathsf{hs}])$
    $\mathsf{sk}'_1[\mathsf{fhe}], \ldots, \mathsf{sk}'_{\mathsf{N}}[\mathsf{fhe}] \leftarrow \mathsf{SS}(\mathsf{d}, \mathsf{N}, \mathsf{sk_{fhe}}, \rho[SS])$
    for $i = 1, \ldots, \mathsf{N}$
        $\mathsf{sk}'_i[\mathsf{hs}] \leftarrow \mathsf{HS.Sign}(\mathsf{sk_{hs}}, \text{“}i\text{”}, \mathsf{sk}'_i[\mathsf{fhe}]; \rho_i[\mathsf{hs}])$
        $\mathsf{sk}'_i := (\mathsf{sk}'_i[\mathsf{fhe}], \mathsf{sk}'_i[\mathsf{hs}])$
    return $\mathsf{sk}'_1, \ldots \mathsf{sk}'_{\mathsf{N}}$

$\mathsf{sUT.PartEval}(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}, \mathsf{ct}_x, \mathsf{ct}_{\mathrm{tpd}}), \mathsf{sk}_i, i, C)$
    $\mathsf{ct}' \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk_{fhe}}, \mathsf{ct}_x, C)$
    $y_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{sk}_i[\mathsf{fhe}], \mathsf{ct}')$
    $\sigma_i \leftarrow \mathsf{HS.Eval}(\mathsf{pk_{hs}}, \text{“}i\text{”}, C_{\mathrm{Dec}}, \mathsf{sk}_i[\mathsf{hs}])$
        where $C_{\mathrm{Dec}} := \mathsf{TFHE.PartDec}(\cdot, \mathsf{ct}')$
    return $(y_i, \sigma_i)$

$\mathsf{sUT.TrapdEval}(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}, \mathsf{ct}_x, \mathsf{ct}_{\mathrm{tpd}}), \mathsf{sk}_i, i, C)$
    $\mathsf{ct}' \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk_{fhe}}, \mathsf{ct}_{\mathrm{tpd}}, C)$
    $y_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{sk}_i[\mathsf{fhe}], \mathsf{ct}')$
    $\sigma_i \leftarrow \mathsf{HS.Eval}(\mathsf{pk_{hs}}, \text{“}i\text{”}, C_{\mathrm{Dec}}, \mathsf{sk}_i[\mathsf{hs}])$
        where $C_{\mathrm{Dec}} := \mathsf{TFHE.PartDec}(\cdot, \mathsf{ct}')$
    return $(y_i, \sigma_i)$

$\mathsf{sUT.VfyEval}\,(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}), \pi_i = (y_i, \sigma_i), i, C)$
    $\mathsf{ct}' \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk_{fhe}}, \mathsf{ct}_x, C)$
    return $\mathsf{HS.Verify}(\mathsf{pk_{hs}}, \text{“}i\text{”}, y_i, \sigma_i, C_{\mathrm{Dec}})$
        where $C_{\mathrm{Dec}} := \mathsf{TFHE.PartDec}(\cdot, \mathsf{ct}')$

$\mathsf{sUT.VfyTrapdEval}\,(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}), \pi_i = (y_i, \sigma_i), i, C)$
    $\mathsf{ct}' \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk_{fhe}}, \mathsf{ct}_{\mathrm{tpd}}, C)$
    return $\mathsf{HS.Verify}(\mathsf{pk_{hs}}, \text{“}i\text{”}, y_i, \sigma_i, C_{\mathrm{Dec}})$
        where $C_{\mathrm{Dec}} := \mathsf{TFHE.PartDec}(\cdot, \mathsf{ct}')$

$\mathsf{sUT.Combine}\,(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}), y_1, \ldots, y_{\mathsf{d}})$
    return $\mathsf{TFHE.Dec}(\mathsf{pk_{fhe}}, \{y_1, \ldots, y_{\mathsf{d}}\})$.
```

Fig. 8: Our sUT Construction.

```
$\underline{\mathsf{pUT.Setup}(1^\lambda, \mathsf{d}, \mathsf{N}, \mathbf{epk}^0, x) \to (\mathsf{pp}, \mathbf{ct})}$
  $(\mathsf{pp}, \mathbf{sk}, \mathsf{trapd}) \leftarrow \mathsf{sUT.Setup}(1^\lambda, \mathsf{d}, \mathsf{N}, x)$
  Parse $\mathsf{trapd}$ as $(\mathsf{sk_{fhe}}, \mathsf{sk_{hs}}, \rho)$
  $\mathbf{ct} \leftarrow \mathcal{F}^{\text{resh}}_{0,\mathbf{epk}^0}(\mathsf{sk_{fhe}}, \mathsf{sk_{hs}}, \rho)$
  return $(\mathsf{pp}, \mathbf{ct})$

$\underline{\mathsf{pUT.Reshare}(\mathsf{sk}^t_i, i, \mathbf{epk}^{t+1}) \to (y^{\text{resh}}_i, \sigma^{\text{resh}}_i)}$
  return $\mathsf{sUT.TrapdEval}(\mathsf{pp}, \mathsf{sk}^t_i, \mathcal{F}^{\text{resh}}_{t,\mathbf{epk}^{t+1}})$

$\underline{\mathsf{pUT.VfyReshare}(\mathsf{pp}, \mathbf{epk}^t, (y^{\text{resh}}_i, \sigma^{\text{resh}}_i), i)}$
  return $\mathsf{sUT.VfyEval}(\mathsf{pp}, (y^{\text{resh}}_i, \sigma^{\text{resh}}_i), i, \mathcal{F}^{\text{resh}}_{t,\mathbf{epk}^{t+1}})$

$\underline{\mathsf{pUT.Reconstruct}(\mathsf{pp}, esk^t_j, (y^{\text{resh}}_i)_{i\in[\mathsf{d}]})}$
  $\mathsf{ct}_j \leftarrow \mathsf{sUT.Combine}(\mathsf{pp}, (y^{\text{resh}}_i)_{i\in[\mathsf{d}]}))$
  return $\mathsf{sk}^{t+1}_j := \mathsf{Dec}_{esk^t_j}(\mathsf{ct}_j)$

$\mathsf{pUT.Eval} := \mathsf{sUT.Eval}$

$\mathsf{pUT.VfyEval} := \mathsf{sUT.VfyEval}$

$\mathsf{pUT.Combine} := \mathsf{sUT.Combine}$
```

```
$\underline{\mathcal{F}^{\text{resh}}_{t,\mathbf{epk}^{t+1}}(s)}$
  Parse $s$ as $s = (\mathsf{sk_{fhe}}, \mathsf{sk_{hs}}, \rho)$
  $(\rho^{t+1}_{\mathsf{SS}}, \rho^{t+1}_{\mathsf{hs},1}, \ldots, \rho^{t+1}_{\mathsf{hs,N}}, \rho^{t+1}_{\mathsf{ct},1}, \ldots, \rho^{t+1}_{\mathsf{ct,N}}) = \mathsf{PRF}_\rho(t+1)$
  $(\mathsf{sk}'_i[\mathsf{fhe}])_{i\in[\mathsf{N}]} \leftarrow \mathsf{SS.Share}(\mathsf{d}, \mathsf{N}, \mathsf{sk_{fhe}}, \rho^{t+1}_{\mathsf{SS}})$
  for $i = 1, \ldots, \mathsf{N}$
    $\mathsf{sk}'_i[\mathsf{hs}] \leftarrow \mathsf{HS.Sign}(\mathsf{sk_{hs}}, \text{“}(i,t)\text{”}, \mathsf{sk}'_i[\mathsf{fhe}], \rho^{t+1}_{\mathsf{hs},i})$
    $\mathsf{sk}'_i := (\mathsf{sk}'_i[\mathsf{fhe}], \mathsf{sk}'_i[\mathsf{hs}])$
    $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{epk^{t+1}_i}(\mathsf{sk}'_i; \rho^{t+1}_{\mathsf{ct},i})$
  return $\mathsf{ct}_1, \ldots \mathsf{ct_N}$
```

Fig. 9: pUT Construction and the auxiliary resharing functionality

References

1. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. pages 2087–2104, 2017.
2. Benny Applebaum. *Cryptography in Constant Parallel Time*. Springer Science & Business Media, 2013.
3. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? pages 260–290, 2020.
4. Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the snark. *Journal of Cryptology*, 30(4):989–1066, 2017.
5. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. pages 326–349, 2012.
6. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. pages 315–333, 2013.
7. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In *International Workshop on Selected Areas in Cryptography*, pages 62–75. Springer, 2002.
8. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. Cryptology ePrint Archive, Report 2017/956, 2017. `https://eprint.iacr.org/2017/956`.
9. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. pages 565–596, 2018.
10. Dan Boneh, Rosario Gennaro, Steven Goldfeder, and Sam Kim. A lattice-based universal thresholdizer for cryptographic systems. *IACR Cryptol. ePrint Arch.*, 2017:251, 2017.
11. Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. pages 108–125, 2008.
12. Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. pages 247–277, 2017.
13. Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. pages 222–255, 2018.
14. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. pages 509–539, 2016.
15. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. pages 21:1–21:21, 2018.
16. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. pages 3–33, 2019.
17. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. pages 423–443, 2020.
18. Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. Cryptology ePrint Archive, 2021. `https://eprint.iacr.org/2021/1423`.

19. Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. pages 229–243, 2017.
20. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. pages 1082–1090, 2019.
21. Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. pages 193–221, 2018.
22. Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. pages 41–59, 2006.
23. George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. pages 532–550, 2014.
24. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. pages 33–62, 2018.
25. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. pages 626–645, 2013.
26. Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. Lattice-based zk-SNARKs from square span programs. pages 556–573, 2018.
27. Craig Gentry. Fully homomorphic encryption using ideal lattices. pages 169–178, 2009.
28. Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. 28(4):820–843, October 2015.
29. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: You only speak once. In *Annual International Cryptology Conference*, pages 64–93. Springer, 2021.
30. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. pages 99–108, 2011.
31. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. pages 218–229, 1987.
32. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. pages 469–477, 2015.
33. Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. *IACR Cryptol. ePrint Arch.*, 2020:504, 2020.
34. Jens Groth. On the size of pairing-based non-interactive arguments. pages 305–326, 2016.
35. Yuval Ishai, Hang Su, and David J. Wu. Shorter and faster post-quantum designated-verifier zksnarks from lattices. Cryptology ePrint Archive, Report 2021/977, 2021. `https://ia.cr/2021/977`.
36. Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential ddh. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–32. Springer, 2021.
37. Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. In *Annual International Cryptology Conference*, pages 89–114. Springer, 2019.
38. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. pages 475–484, 2014.

Supporting Material

A Additional Preliminaries

A.1 Pseudorandom Functions

We recall the definition of pseudorandom functions.

Definition 8 (PRF). *Let* $\mathsf{PRF}\colon \mathcal{K} \times \mathsf{D} \to \mathsf{R}$ *be a family of functions and let* Γ *be the set of all functions* $\mathsf{D} \to \mathsf{R}$*. We say that* PRF *is a pseudorandom function (PRF) (family) if it is efficiently computable and for all PPT distinguishers* $\mathcal{D}$

$$\left|\Pr\left[\mathsf{K} \leftarrow\!\!\$\, \mathcal{K}, \mathcal{D}^{\mathsf{PRF}_{\mathsf{K}}(\cdot)}(1^{\lambda})\right] - \Pr\left[g \leftarrow\!\!\$\, \Gamma, \mathcal{D}^{g(\cdot)}(1^{\lambda})\right]\right| \leq \mathsf{negl}(\lambda)$$

A.2 Homomorphic Signatures

A homomorphic signature scheme allows a party P_1 to sign a message x using a secret signing key $\mathsf{sk_{hs}}$ and distribute this signature to some untrusted party P_2 who can perform arbitrary computations $y = f(x)$ over x and homomorphically derive a signature $\sigma_{f,y}$. This signature certifies that y is the correct output of the computation f over x. The signature $\sigma_{f,y}$ is required to have a short length which is also independent of the size of x. Moreover, anyone should be able to verify publicly the tuple $(f, y, \sigma_{f,y})$ using P_1's public verification key $\mathsf{vk_{hs}}$ and become convinced that y is indeed $f(x)$, without needing to have x.

Formally, a homomorphic signature scheme is as follows:

Definition 9. *A homomorphic signature scheme* HS *for a function family* $\mathfrak{F}$ *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Eval})$ *defined as follows:*

$\mathsf{HS.Setup}(1^{\lambda}) \to (\mathsf{sk_{hs}}, \mathsf{pk_{hs}})$**:** *The setup algorithm takes a security parameter* λ *and outputs a secret key* $\mathsf{sk_{hs}}$ *and a public key* $\mathsf{pk_{hs}}$*. We assume that the public key defines a set* $\mathfrak{F}$ *of "admissible" functions.*

$\mathsf{HS.Sign}(\mathsf{sk_{hs}}, \tau, m) \to \sigma$**:** *The signing algorithm takes a secret key* $\mathsf{sk_{hs}}$*, a tag* τ *and a message* m*, and outputs a signature* σ*.*

$\mathsf{HS.Verify}(\mathsf{pk_{hs}}, \tau, m, \sigma, f) \to \{0, 1\}$**:** *The verification algorithm takes a public key* $\mathsf{pk_{hs}}$*, a tag* τ*, a message* m*, a signature* σ *and a function* $f \in \mathfrak{F}$*, and returns a bit indicating accept or reject.*

$\mathsf{HS.Eval}(\mathsf{pk_{hs}}, \tau, f, \sigma) \to \sigma_{f,y}$**:** *The evaluation algorithm takes a public key* $\mathsf{pk_{hs}}$*, a tag* τ*, a function* $f \in \mathfrak{F}$ *and a signature* σ*, and outputs a signature* $\sigma_{f,y}$*.*

A HS scheme should satisfy the following properties:

– **Correctness.** We require that for any $(\mathsf{sk_{hs}}, \mathsf{pk_{hs}}) \leftarrow \mathsf{Setup}(1^{\lambda})$, it holds that

1. For all tags τ and all m,

$$\Pr\left[\sigma \leftarrow \mathsf{HS.Sign}(\mathsf{sk}_{\mathsf{hs}}, \tau, m) \ : \ \mathsf{HS.Verify}(\mathsf{pk}_{\mathsf{hs}}, \tau, m, \sigma, I) = 1\right] = 1 - \mathsf{negl}(\lambda)$$

   where I is the identity function.
2. For all tags τ, all m and all $f \in \mathfrak{F}$,

$$\Pr\left[\begin{matrix} \sigma \leftarrow \mathsf{HS.Sign}(\mathsf{sk}_{\mathsf{hs}}, \tau, m) \\ \sigma_{f,y} \leftarrow \mathsf{HS.Eval}(\mathsf{pk}_{\mathsf{hs}}, \tau, f, \sigma) \end{matrix} \ : \ \mathsf{HS.Verify}(\mathsf{pk}_{\mathsf{hs}}, \tau, f(m), \sigma_{f,y}, f) = 1\right] = 1 - \mathsf{negl}(\lambda)$$

– **Unforgeability** Given a homomorphically signed data m an adversary cannot produce a function f and a valid signature $\sigma_{y'}$ for which $f(m) \neq y'$ (for a formal definition we refer the reader to Definition 3.2 in [10]).
– **Context-hiding.** There exists a simulator such that for all $\lambda, \tau, x, f \in \mathfrak{F}$ we have that

$$S(\mathsf{sk}_{\mathsf{hs}}, \tau, f, f(x)) \approx \mathsf{HS.Eval}(\mathsf{pk}_{\mathsf{hs}}, \tau, f, \sigma_x)$$

  where $(\mathsf{sk}_{\mathsf{hs}}, \mathsf{pk}_{\mathsf{hs}}) \leftarrow \mathsf{Setup}(1^\lambda)$ and $\sigma_x \leftarrow \mathsf{HS.Sign}(\mathsf{sk}_{\mathsf{hs}}, \tau, m)$
– **Compactness.** The output of a $\mathsf{HS.Eval}$ should be of size linear in the depth of the circuit representing f.

A.3 Threshold Fully Homomorphic Encryption

In this section, we recall the syntax and basic security notions of (leveled) threshold fully homomorphic encryption (TFHE) from [8]. It is presented for general access structures although we just require (d, N) threshold structures in this work. The scheme is leveled as it requires correctness only for circuits of bounded depth.

Definition 10. *((Leveled) Threshold Fully Homomorphic Encryption (TFHE)) Let $P = \{P_1, \ldots, P_{\mathsf{N}}\}$ be a set of parties and let $\mathbb{S}$ be a class of efficient access structures on P. A threshold fully encryption scheme for $\mathbb{S}$ is a tuple of PPT algorithms $\mathsf{TFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PartDec}, \mathsf{Dec})$ defined as follows:*

– $\mathsf{TFHE.Setup}(1^\lambda, \mathbb{A}) \rightarrow (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_{\mathsf{N}})$*: The setup algorithm inputs the security parameter λ and an access structure $\mathbb{A}$, and outputs a public key* pk *and a set of key shares* $\mathsf{sk}_1, \ldots, \mathsf{sk}_{\mathsf{N}}$.
– $\mathsf{TFHE.Enc}(\mathsf{pk}, m) \rightarrow ct$*: The encryption algorithm inputs a public key* pk *and a message m, and outputs a ciphertext ct.*
– $\mathsf{TFHE.Eval}(\mathsf{pk}, C, ct) \rightarrow \hat{ct}$*: The evaluation algorithm inputs a public key* pk*, a circuit $C : \{0,1\}^* \rightarrow \{0,1\}^*$, and a ciphertext ct, and outputs a ciphertext $\hat{ct}$.*
– $\mathsf{TFHE.PartDec}(\mathsf{pk}, ct, \mathsf{sk}_i) \rightarrow m_i$*: The partial decryption algorithm inputs a public key* pk*, a ciphertext ct and a secret key share* sk_i*, and outputs a partial decryption m_i related to party P_i.*
– $\mathsf{TFHE.Dec}(\mathsf{pk}, B) \rightarrow \hat{m}$*: The final decryption algorithm inputs a public key* pk*, a set $B = \{m_i\}_{i \in S}$ for some $S \subset \{P_1, \ldots, P_{\mathsf{N}}\}$, and outputs a message $\hat{m}$ or $\perp$.*

We require the following properties for a TFHE scheme. See aforementioned papers for further formal details.

– **Evaluation Correctness.** For all λ, access structure $\mathbb{A}$, circuit $C : \{0,1\}^k \to \{0,1\}^*$ of depth at most $\delta = \mathsf{poly}(\lambda)$, $S \in \mathbb{A}$, and message $m \in \{0,1\}^k$,

$$\Pr\left[\begin{array}{r}(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, \mathbb{A}) \\ ct \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, m) \\ \hat{ct} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk}, C, ct) \\ B \leftarrow \{\mathsf{TFHE.PartDec}(\mathsf{pk}, \hat{ct}, \mathsf{sk}_i)\}_{i \in S}\end{array} : \mathsf{TFHE.Dec}(\mathsf{pk}, B) \to C(m)\right] = 1 - \mathsf{negl}(\lambda)$$

– **Setup with Explicit Secret Sharing.** We require a special structural property for the setup algorithm of TFHE. This will be useful in our construction. We point out that this assumptions is satisfied by natural constructions such as those in [8]. More formally we require the TFHE key generation to output $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{pk}_\mathsf{fhe})$ where $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})$ are (d, N) secret shares of sk_fhe, that is $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{Share}^{\mathsf{d},\mathsf{N}}(\mathsf{sk})$ and we let $(\mathsf{sk}_\mathsf{fhe}, \mathsf{pk}_\mathsf{fhe}) \leftarrow \mathsf{TFHE.KeyGenAux}(1^\lambda, \mathsf{d}, \mathsf{N})$ for an auxiliary function $\mathsf{TFHE.KeyGenAux}$.

– **Semantic Security.** This is the standard requirement of semantic security for public-key encryption. We say that a TFHE scheme satisfies semantic security if for all λ, for all PPT adversary $\mathcal{A}$, and any circuit C of depth at most $\delta = \mathsf{poly}(\lambda)$, the following experiment $\mathsf{Exp}^{\mathsf{sem}}_{\mathcal{A},\mathsf{TFHE}}(1^\lambda)$ outputs 1 with negligible probability.

```
$\mathsf{Exp}^{\mathsf{sem}}_{\mathcal{A},\mathsf{TFHE}}(1^\lambda)$
    $\mathbb{A} \leftarrow \mathcal{A}(1^\lambda, C)$; **if** $\mathbb{A} \notin \mathbb{S}$, **then return** $0$;
    $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, \mathbb{A})$;
    $S \leftarrow \mathcal{A}(\mathsf{pk})$; **if** $S \not\subseteq \{P_1, \ldots, P_\mathsf{N}\} \vee S \in \mathbb{A}$, **then return** $0$;
    $b \leftarrow\!\!\$\ \{0,1\}$; $ct \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, b)$;
    $b' \leftarrow \mathcal{A}(ct, \{\mathsf{sk}_i\}_{i \in S})$; **if** $b = b'$, **then return** $1$
```

– **KDM (circular) Security.** We require a specific type of KDM (Key-Dependent Message) security as defined in [7, 11] where the adversary cannot distinguish between an encryption of the secret key and an encryption of an arbitrary message (here we use the string of all 0s but, together with semantic security, this implies that an encryption of the secret key is indistinguishable from any other message). We use the following game between a challenger and an adversary $\mathcal{A}$. For a security parameter λ and all threshold access structures (d, N) the game proceeds as follows: The challenger chooses a random bit $b \leftarrow\!\!\$\ \{0,1\}$. It generates a key $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TFHE.KeyGenAux}(1^\lambda, \mathsf{d}, \mathsf{N})$ and sends pk to the adversary as well as c such that $c \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, \mathsf{sk})$ if $b = 0$, otherwise $c \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, 0^{|\mathsf{sk}|})$. The adversary outputs a bit b'. The property is satisfied if $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

– **Simulation Security.** This notion corresponds to a specialized notion similar to that of sUT security: we require that there exists a pair of simulators S_1, S_2—respectively for the setup of the scheme and for partial decryption queries—such that no adversary can distinguish whether it is interacting with a real or simulated partial decryption oracle. More formally: we say that a TFHE scheme satisfies simulation security if for all λ and access structure $\mathbb{A}$, there exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for any PPT adversary $\mathcal{A}$, and any circuit C of depth at most $\delta = \mathsf{poly}(\lambda)$, the following two experiments $\mathsf{Exp}_{\mathcal{A},\mathsf{Real}}(1^\lambda)$ and $\mathsf{Exp}_{\mathcal{A},\mathsf{Ideal}}(1^\lambda)$ are indistinguishable. Note that when defining the oracles, we implicitly assume that the inputs are valid, i.e., for input (S, C), the oracles return $\perp$ if $S \not\subseteq \{P_1, \ldots, P_\mathsf{N}\}$ or C is not of polynomial depth.

```
$\mathsf{Exp}_{\mathcal{A},\mathsf{Real}}(1^\lambda)$
  $\mathbb{A} \leftarrow \mathcal{A}(1^\lambda, C)$; if $\mathbb{A} \notin \mathbb{S}$, then return $0$;
  $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, \mathbb{A})$;
  $(S^*, m_1, \ldots, m_k) \leftarrow \mathcal{A}(\mathsf{pk})$;
if $S \not\subseteq \{P_1, \ldots, P_\mathsf{N}\} \vee S \in \mathbb{A}$, then return $0$;
  $\{ct_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, m_i)\}_{i \in [k]}$;
  $b \leftarrow \mathcal{A}^{\mathcal{O}\text{-}real}(\{ct_i\}_{i \in [k]}, \{\mathsf{sk}_i\}_{i \in S^*})$;

$\mathcal{O}\text{-}real(S, C)$
  $\hat{ct} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk}, C, ct_1, \ldots, ct_k)$;
  $B \leftarrow \{\mathsf{TFHE.PartDec}(\mathsf{pk}, \hat{ct}, \mathsf{sk}_i)\}_{i \in S}$;
  return $B$;

$\mathsf{Exp}_{\mathcal{A},\mathsf{Ideal}}(1^\lambda)$
  $\mathbb{A} \leftarrow \mathcal{A}(1^\lambda, C)$; if $\mathbb{A} \notin \mathbb{S}$, then return $0$;
  $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{state}) \leftarrow \mathcal{S}_1(1^\lambda, \mathbb{A})$;
  $(S^*, m_1, \ldots, m_k) \leftarrow \mathcal{A}(\mathsf{pk})$;
if $S \not\subseteq \{P_1, \ldots, P_\mathsf{N}\} \vee S \in \mathbb{A}$, then return $0$;
  $\{ct_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, m_i)\}_{i \in [k]}$;
  $b \leftarrow \mathcal{A}^{\mathcal{O}\text{-}ideal}(\{ct_i\}_{i \in [k]}, \{\mathsf{sk}_i\}_{i \in S^*})$;

$\mathcal{O}\text{-}ideal$
  $y \leftarrow C(m_1, \ldots, m_k)$;
  $B \leftarrow \mathcal{S}_2(C, \{ct_1, \ldots, ct_k\}, y, S, \mathsf{state})$;
  return $B$;
```

Fig. 10: Simulation Security Experiments

Remark 4 (Constructions of TFHE). The work in [8] constructs TFHE with the special setup structure above from secret-sharing and lattices (LWE). They build FHE from the same assumptions. In our work we make the (mild) assumption on KDM security of their scheme.

A.4 Homomorphic Secret Sharing

The following is an adaptation of the definition of HSS from [14]; our notion is non-additive.

Definition 11 (Homomorphic Secret Sharing (HSS) [14]). *A (2-party) HSS scheme is a quadruple* $(\mathsf{Share}, \mathsf{Eval}_1, \mathsf{Eval}_2, \mathsf{Combine})$ *such that:*

$\mathsf{Share}(1^\lambda, x) \to (\mathsf{share}_1, \mathsf{share}_2)$ *produces two secret shares of a secret x.*
$\mathsf{Eval}_b(\mathsf{share}_b, C) \to y_b$ *produces a share of the evaluation $C(x)$ given share* share_b *for $b \in \{1,2\}$ of x and circuit C.*
$\mathsf{Combine}(y_1, y_2) \to y$ *it produces a final output $y \in \{0,1\}^*$ on input a pair of output shares y_1, y_2*

The algorithms should satisfy the following correctness and security requirements:

– **Correctness** For any $\lambda \in \mathbb{N}$, $x \in \{0,1\}^*$

$$\Pr\left[\begin{matrix}(\mathsf{share}_1, \mathsf{share}_2) \leftarrow \mathsf{Share}\left(1^\lambda, x\right) \\ y_b \leftarrow \mathsf{Eval}_b\left(\mathsf{share}_b, C\right), b = 1,2\end{matrix} : \mathsf{Combine}(y_1, y_2) = C\left(x\right)\right] \geq 1 - \mathsf{negl}(\lambda)$$

– **Security** For all security parameters $\lambda \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$ we require the following advantage to be negligible in λ:

$$\mathrm{Adv}_{\mathrm{HSS},\mathcal{A}} := \left|\Pr\left[\mathcal{A}(\mathsf{share}_\beta) = \beta \ : \ \begin{matrix} (b, x_0, x_1) \leftarrow \mathcal{A}(1^\lambda) \\ \beta \leftarrow\!\$ \{0,1\} \\ (\mathsf{share}_1, \mathsf{share}_2) \leftarrow \mathsf{Share}(1^\lambda, x_\beta)\end{matrix}\right] - \frac{1}{2}\right|$$

B Details on UT

B.1 Model (from [9])

Universal thresholdizer (UT) generalizes of non-interactive threshold schemes, where for a given a secret known at setup time, one can secret share the secret and now the parties with the share can together produce evaluations of that secret. For simplicity, the formal definition below is tailored to (d, N)-threshold access structure, because our construction is in this setting. The definition can be modified in a straightforward way to consider general access structures.

Definition 12 (Universal Thresholdizer). *Let $P = \{P_1, \ldots, P_\mathsf{N}\}$ be a set of parties. A universal thresholdizer scheme for a* (d, N)*-threshold access structure is a quadruple of PPT algorithms* $\mathsf{UT} = (\mathsf{Setup}, \mathsf{PartEval}, \mathsf{VfyEval}, \mathsf{Combine})$ *with the following properties:*

– $\mathsf{UT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \to (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})$: *On input the security parameter 1^λ, threshold parameters* (d, N)*, and a message $x \in \{0,1\}^k$, the setup algorithm outputs the public parameters* pp*, a set of shares* $\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}$.
– $\mathsf{UT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C) \to \pi_i$: *On input the public parameters* pp*, a share* sk_i*, a tag i, and a circuit $C : \{0,1\}^k \to \{0,1\}$, the evaluation algorithm outputs a partial evaluation π_i.*

– $\mathsf{UT.VfyEval}(\mathsf{pp}, \pi_i, i, C) \to \{0,1\}$ *: On input the public parameters* pp*, a partial evaluation* π_i*, a tag* i*, and a circuit* $C : \{0,1\}^k \to \{0,1\}^*$*, the verification algorithm returns a bit indicating acceptance or rejection.*
– $\mathsf{UT.Combine}(\mathsf{pp}, B) \to \mathsf{y}$ *: On input* pp*, a set of partial evaluations* $B = \{\pi_i\}_i$ *of size* d*, the combining algorithm outputs the final evaluation* y*.*

A UT scheme is required to satisfy *correctness*, *compactness*, *robustness* and *security.*

Definition 13 (Evaluation Correctness). *A* (d, N)-UT *scheme has evaluation correctness if for all* λ*, message* $x \in \{0,1\}^k$*, and circuit* $C : \{0,1\}^k \to \{0,1\}^*$*,*

$$\Pr\left[\begin{array}{r} (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{UT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \\ B \leftarrow \{\mathsf{UT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C)\}_i \end{array} : \begin{array}{r} |B| \geq \mathsf{d} \quad \wedge \\ \mathsf{UT.Combine}(\mathsf{pp}, B) = C(x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Definition 14 (Verification Correctness). *A* (d, N)-UT *scheme has verification correctness if for all* λ*, message* $x \in \{0,1\}^k$*, and circuit* $C : \{0,1\}^k \to \{0,1\}^*$*,*

$$\Pr\left[\begin{array}{r} (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{UT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \\ \pi_i \leftarrow \mathsf{UT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C) \end{array} : \mathsf{UT.VfyEval}(\mathsf{pp}, \pi_i, i, C) = 1\right] = 1$$

Definition 15 (Compactness). *A* UT *scheme is called compact if there exists some polynomial* $\mathsf{poly}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, circuit* $C : \{0,1\}^k \to \{0,1\}$*,*

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{UT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \\ \pi_i \leftarrow \mathsf{UT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C) \ \ \textit{for } i \in [\mathsf{N}] \end{array} : |\pi_i| \leq \mathsf{poly}(\lambda, \mathsf{N})\right] = 1$$

Definition 16 (Robustness). *A* UT *scheme satisfies robustness if for all* $\lambda \in \mathbb{N}$*, it holds that for any PPT adversary* $\mathcal{A}$*, the following experiment* $\mathsf{Expt}_{\mathcal{A},\mathsf{robust}}(1^\lambda)$ *outputs 1 with negligible probability.*

$\underline{\mathsf{ExptRobust}_{\mathcal{A},\mathsf{UT}}(1^\lambda)}$:

1. $\mathcal{A}$ takes in the security parameter 1^λ and outputs a message x.
2. The challenger runs $(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}) \leftarrow \mathsf{UT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right)$ and sends $(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a fake partial evaluation π_i^*.
4. The challenger returns 1 if
   $\pi_i^* \neq \mathsf{UT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C)$ and $\mathsf{UT.VfyEval}(\mathsf{pp}, \pi_i^*, i, C) = 1$.

Definition 17 (UT Security). *A* UT *scheme satisfies security with respect to circuit sampler D if for all λ, the following holds. There exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_S, \mathcal{S}_E)$ such that for any PPT adversary $\mathcal{A}$, the following experiments* $\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{UT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ *and* $\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{UT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ *are computationally indistinguishable.*

$\underline{\mathsf{Expt}^{\mathsf{world}\in\{\mathsf{real},\mathsf{ideal}\}}_{\mathcal{A},\mathsf{UT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right):}$

1. $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{d}, \mathsf{N})$
2. if $\mathsf{world} = \mathsf{real}$ then $(\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}) \leftarrow \mathsf{UT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right)$
3. else if $\mathsf{world} = \mathsf{ideal}$ then $(\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}) \leftarrow \mathcal{S}_S\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$.
4. $\mathcal{A}$ is given (pp) and outputs a corruption set $\mathcal{C}$ of size $t-1$
5. The challenger provides the shares $\{\mathsf{s}_i\}_{i\in\mathcal{C}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ can ask for a polynomial number of adaptive queries to the oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{UT}}$ (defined below).
7. Adversary outputs a guess guess
8. Return 1 iff $\mathsf{guess} = \mathsf{world}$

We now define oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{UT}}$. It can receive as a query C where C is a circuit and the oracle responds with:

– if $\mathsf{world} = \mathsf{real}$ then returns $\{\mathsf{y}_i \leftarrow \mathsf{UT.Eval}(\mathsf{pp}, \mathsf{s}_{i,T}, C)\}_{i\in[\mathsf{N}]}$
– if $\mathsf{world} = \mathsf{ideal}$ then return $\{\mathsf{y}_i\}_{i\in[\mathsf{N}]} \leftarrow \mathcal{S}_E(\mathsf{trapd}, C, C(x))$

B.2 Construction of UT from TFHE and Homomorphic Signatures

The construction in Figure 11 is a formal description of the construction for UT informally discussed in [9]. Its security can be seen as a special case of the formal proof of security of our sUT construction, also based on TFHE and HS.

B.3 Construction of (2-party) UT from HSS and HS

Our construction of UT from HSS and HS is in Figure 6. Given the similarity of syntax and properties between HSS and TFHE (notice that the security of HSS can be cast in terms of TFHE's simulation-based security), the security of the construction in Figure 6 follows with little variations as a special case of the proof of security of our sUT construction (see also Theorem 6).

C Details on Special UT (sUT)

C.1 Model

Universal thresholdizer (UT) is a generalization of non-interactive threshold schemes, where for a given a secret known at setup time, one can secret share the

```
$\mathsf{UT.Setup}(1^\lambda, \mathsf{d}, \mathsf{N}, x)$
    $(\mathsf{sk_{hs}}, \mathsf{pk_{hs}}) \leftarrow \mathsf{HS.Setup}(1^\lambda)$
    $(\mathsf{sk_{fhe}}, \mathsf{pk_{fhe}}) \leftarrow \mathsf{TFHE.KeyGenAux}(1^\lambda, \mathsf{d}, \mathsf{N})$
    $\mathsf{ct}_x \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk_{fhe}}, x)$
    $\mathbf{sk} \leftarrow \mathsf{UT\text{-}AuxSetup}(\mathsf{sk_{fhe}}, \mathsf{sk_{hs}})$
    return $(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}, \mathsf{ct}_x), \mathbf{sk})$

$\mathsf{UT\text{-}AuxSetup}(\mathsf{sk_{fhe}}, \mathsf{sk_{hs}})$
    for $i = 1, \ldots, \mathsf{N}$
        $\mathsf{sk}'_i[\mathsf{fhe}] \leftarrow \mathsf{SS}(\mathsf{d}, \mathsf{N}, \mathsf{sk_{fhe}})$
        $\mathsf{sk}'_i[\mathsf{hs}] \leftarrow \mathsf{HS.Sign}(\mathsf{sk_{hs}}, \text{“}i\text{”}, \mathsf{sk}'_i[\mathsf{fhe}])$
        $\mathsf{sk}'_i := (\mathsf{sk}'_i[\mathsf{fhe}], \mathsf{sk}'_i[\mathsf{hs}])$
    return $\mathsf{sk}'_1, \ldots \mathsf{sk}'_\mathsf{N}$

$\mathsf{UT.PartEval}(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}, \mathsf{ct}_x), \mathsf{sk}_i, i, C)$
    $\mathsf{ct}' \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk_{fhe}}, \mathsf{ct}_x, C)$
    $y_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{sk}_i[\mathsf{fhe}], \mathsf{ct}')$
    $\sigma_i \leftarrow \mathsf{HS.Eval}(\mathsf{pk_{hs}}, \text{“}i\text{”}, C_{\mathrm{Dec}}, \mathsf{sk}_i[\mathsf{hs}])$
        where $C_{\mathrm{Dec}} := \mathsf{TFHE.PartDec}(\cdot, \mathsf{ct}')$
    return $(y_i, \sigma_i)$

$\mathsf{UT.VfyEval}\,(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}), \pi_i = (y_i, \sigma_i), i, C)$
    $\mathsf{ct}' \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk_{fhe}}, \mathsf{ct}_x, C)$
    return $\mathsf{HS.Verify}(\mathsf{pk_{hs}}, \text{“}i\text{”}, y_i, \sigma_i, C_{\mathrm{Dec}})$
        where $C_{\mathrm{Dec}} := \mathsf{TFHE.PartDec}(\cdot, \mathsf{ct}')$

$\mathsf{UT.Combine}\,(\mathsf{pp} := (\mathsf{pk_{fhe}}, \mathsf{pk_{hs}}), y_1, \ldots, y_\mathsf{d})$
    return $\mathsf{TFHE.Dec}(\mathsf{pk_{fhe}}, \{y_1, \ldots, y_\mathsf{d}\})$.
```

Fig. 11: UT Construction from TFHE and HS (informally described in [9]).

secret and now the parties with the share can together produce evaluations of that secret. We define a special type of UT called sUT which essentially is a UT with two extra algorithms that allow (verification of) evaluations on its own trapdoor. For simplicity, the formal definition below is tailored to (d, N)-threshold access structure, because our construction is in this setting. The definition can be modified in a straightforward way to consider general access structures.

Definition 18 (Special Universal Thresholdizer). *Let $P = \{P_1, \ldots, P_\mathsf{N}\}$ be a set of parties. A universal thresholdizer scheme for a (d, N)-threshold access structure is a quadruple of PPT algorithms* $\mathsf{sUT} = (\mathsf{Setup}, \mathsf{PartEval}, \mathsf{VfyEval}, \mathsf{TrapdEval}, \mathsf{VfyTrapdEval}, \mathsf{Combine})$ *with the following properties:*

- $\mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \rightarrow (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{trapd})$: *On input the security parameter 1^λ, threshold parameters (d, N), and a message $x \in \{0,1\}^k$, the setup algorithm outputs the public parameters* pp*, a set of shares* $\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}$*, and a trapdoor* trapd.
- $\mathsf{sUT.PartEval}\,(\mathsf{pp}, \mathsf{sk}_i, i, C) \rightarrow \pi_i$: *On input the public parameters* pp*, a share* sk_i*, a tag i, and a circuit $C : \{0,1\}^k \rightarrow \{0,1\}$, the evaluation algorithm outputs a partial evaluation π_i.*

– $\mathsf{sUT.VfyEval}(\mathsf{pp}, \pi_i, i, C) \to \{0,1\}$: *On input the public parameters* pp*, a partial evaluation* π_i*, a tag* i*, and a circuit* $C : \{0,1\}^k \to \{0,1\}^*$*, the verification algorithm returns a bit indicating acceptance or rejection.*
– $\mathsf{sUT.TrapdEval}(\mathsf{pp}, \mathsf{sk}_i, i, C) \to \pi_i$ *: On input* pp*, a share* sk_i*, a tag* i*, and a circuit* $C : \{0,1\}^k \to \{0,1\}$*, the trapdoor evaluation algorithm outputs a trapdoor partial evaluation* π_i*.*
– $\mathsf{sUT.VfyTrapdEval}(\mathsf{pp}, \pi_i, i, C) \to \{0,1\}$ *: On input the public parameters* pp*, a trapdoor partial evaluation* π_i*, a tag* i*, and a circuit* $C : \{0,1\}^k \to \{0,1\}^*$*, the trapdoor verification algorithm outputs a bit indicating acceptance or rejection.*
– $\mathsf{sUT.Combine}(\mathsf{pp}, B) \to \mathsf{y}$ *: On input* pp*, a set of partial evaluations* $B = \{\pi_i\}_i$ *of size* d*, the combining algorithm outputs the final evaluation* y*.*

A sUT scheme is required to satisfy *correctness*, *compactness*, *robustness* and *security.*

Definition 19 (Evaluation Correctness). *A* (d, N)-sUT *scheme has evaluation correctness if for all* λ*, message* $x \in \{0,1\}^k$*, and circuit* $C : \{0,1\}^k \to \{0,1\}^*$*,*

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{trapd}) \\ \quad \leftarrow \mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \\ B \leftarrow \{\mathsf{sUT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C)\}_i\end{array} : \begin{array}{c}|B| \geq \mathsf{d} \quad \wedge \\ \mathsf{sUT.Combine}(\mathsf{pp}, B) = C(x)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

A similar property holds for a valid set of trapdoor partial evaluations (computed by $\mathsf{TrapdEval}$ *algorithm) as well.*

Definition 20 (Verification Correctness). *A* (d, N)-sUT *scheme has verification correctness if for all* λ*, message* $x \in \{0,1\}^k$*, and circuit* $C : \{0,1\}^k \to \{0,1\}^*$*,*

$$\Pr\left[\begin{array}{r}(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \\ \pi_i \leftarrow \mathsf{sUT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C)\end{array} : \mathsf{sUT.VfyEval}(\mathsf{pp}, \pi_i, i, C) = 1\right] = 1$$

A similar property holds for the $\mathsf{TrapdEval}$ *and* $\mathsf{VfyTrapdEval}$ *algorithms as well.*

Definition 21 (Compactness). *A* sUT *scheme is called compact if there exists some polynomial* $\mathsf{poly}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, circuit* $C : \{0,1\}^k \to \{0,1\}$*,*

$$\Pr\left[\begin{array}{c}(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right) \\ \pi_i \leftarrow \mathsf{sUT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C) \;\; \textit{for } i \in [\mathsf{N}]\end{array} : |\pi_i| \leq \mathsf{poly}(\lambda, \mathsf{N})\right] = 1$$

Definition 22 (Robustness). *A* sUT *scheme satisfies robustness if for all* $\lambda \in \mathbb{N}$*, it holds that for any PPT adversary* $\mathcal{A}$*, the following experiment* $\mathsf{Expt}_{\mathcal{A},\mathsf{robust}}(1^\lambda)$ *outputs 1 with negligible probability.*

$\underline{\mathsf{ExptRobust}_{\mathcal{A},\mathsf{sUT}}(1^\lambda):}$

1. $\mathcal{A}$ takes in the security parameter 1^λ and outputs a message x.
2. The challenger runs $(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right)$ and sends $(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a fake partial evaluation π_i^*.
4. The challenger returns 1 if $\pi_i^* \neq \mathsf{sUT.PartEval}(\mathsf{pp}, \mathsf{sk}_i, i, C)$ and $\mathsf{sUT.VfyEval}(\mathsf{pp}, \pi_i^*, i, C) = 1$.

Security. The security of a special-evaluation sUT scheme is a modified version of the security property of UT schemes. In standard UT we allow the adversary to access evaluation queries only to a secret x provided by the adversary itself. In this other version we will allow the adversary to access evaluation queries to a trapdoor of the UT setup itself. Naturally, for this notion to allow security we restrict the type of queries the adversary can access to only those coming from a "benign" distribution (otherwise the adversary could, for example, ask the identity function itself and thus break security).

Definition 23 (Circuit Sampler). *A circuit sampler D is a PPT that on input a string z returns a circuit $C \leftarrow D(z)$ of size polynomial in $|z|$.*

Definition 24 (sUT Security). *A* sUT *scheme satisfies security with respect to circuit sampler D if for all λ, the following holds. There exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_S, \mathcal{S}_E)$ such that for any PPT adversary $\mathcal{A}$, the following experiments $\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ and $\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ are computationally indistinguishable.*

We now define oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{sUT}}$ for the experiment in Figure 12. It can receive in input either a tuple $(\mathsf{trapdquery}, z)$ or a tuple (xquery, C). The first asks for a query evaluation on the trapdoor; the other for the secret x. For the case $(\mathsf{trapdquery}, z)$, the oracle samples a circuit from sampler D as $C \leftarrow\!\!\$\ D(\mathsf{trapd}, z)$, returns circuit C and partial evaluation $\{\mathsf{y}_i \leftarrow \mathsf{sUT.TrapdEval}(\mathsf{pp}, \mathsf{s}_i, C)\}_{i\in[\mathsf{N}]}$.

For the case (xquery, C) the oracle responds with:

– if $\mathsf{world} = \mathsf{real}$ then return $\{\mathsf{y}_i \leftarrow \mathsf{sUT.Eval}(\mathsf{pp}, \mathsf{s}_{i,T}, C)\}_{i\in[\mathsf{N}]}$
– if $\mathsf{world} = \mathsf{ideal}$ then return $\{\mathsf{y}_i\}_{i\in[\mathsf{N}]} \leftarrow \mathcal{S}_E(\mathsf{trapd}, C, C(x))$

C.2 Construction of sUT

We extend the construction from [9] by replacing NIZKs with homomorphic signatures and to support partial evaluations on the sUT trapdoor. The TrapdEval function works exactly as PartEval but one a different ciphertext (which encrypts the trapdoor). The construction is in Figure 8.
The construction uses the following building blocks:

$\underline{\mathsf{Expt}^{\mathsf{world}\in\{\mathsf{real},\mathsf{ideal}\}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda,\mathsf{d},\mathsf{N}\right):}$

1. $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{d}, \mathsf{N})$
2. if $\mathsf{world} = \mathsf{real}$ then $(\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathsf{sUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right)$
3. else if $\mathsf{world} = \mathsf{ideal}$ then $(\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{N}, \mathsf{trapd}) \leftarrow \mathcal{S}_S\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$.
4. $\mathcal{A}$ is given (pp) and outputs a corruption set $\mathcal{C}$ of size $t-1$
5. The challenger provides the shares $\{\mathsf{s}_i\}_{i\in\mathcal{C}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ can ask for a polynomial number of adaptive queries to the oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{sUT}}$ (defined below).
7. Adversary outputs a guess guess
8. Return 1 iff $\mathsf{guess} = \mathsf{world}$

Fig. 12: sUT security experiment. We mark in blue the few differences with the UT security experiment.

– a TFHE scheme (Definition 10). We require two things on it. First we require the secret keys for partial evaluations to be a secret share of a secret (this property is called "Setup with Explicit Secret Sharing" in Definition 10). More formally we require the TFHE key generation to output $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}, \mathsf{pk}_\mathsf{fhe})$ where $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})$ are (d, N) secret shares of sk_fhe and we let $(\mathsf{sk}_\mathsf{fhe}, \mathsf{pk}_\mathsf{fhe}) \leftarrow \mathsf{TFHE.KeyGenAux}(1^\lambda, \mathsf{d}, \mathsf{N})$ for an auxiliary function $\mathsf{TFHE.KeyGenAux}$. This will be necessary later for correctness of the proactive UT we build on top of our sUT. We remark that the constructions in [9] satisfy these notions. Second, we require the TFHE to have Key Dependent Message security (Definition 7.1 in [7]). This is necessary because we will encrypt sk_fhe as defined above with its corresponding public key. This requirement, although heuristic, has been made before in the context of homomorphic encryption [27].
– Context-hiding homomorphic signatures (Definition 9)

The construction in Figure 8 has an auxiliary Setup function used to make clearer the symmetry with the resharing function we will define later.

We prove security of sUT with respect to distributions related to a "resharing" function (defined below). We note, however, that one can prove security for other distributions and circuits as long as they do not leak the secret key for TFHE to the adversary. We leave modeling this general case as future work. This resharing function will be used later for pUT and its motivation can be better understood from the pUT section. Nonetheless we define it here because we will prove sUT security with respect to it. The superscripts $t+1$ refer to the fact that these values are (or are used for) part of the secret shares of the committee members for epoch $t+1$.

Definition 25 (Resharing Circuit Sampler). *Given parameters* d, N *and a public key encryption scheme* PK*, the resharing circuit sampler* $D'(z)$ *is defined as follows:*

– *Parse* z *as* $(t, (\hat{\mathsf{epk}}_i)_{i\in[\mathsf{d}-1]})$
– *Sample remaining* $\mathsf{N} - \mathsf{d}$ *ephemeral keys for* $i = \mathsf{d}, \ldots, \mathsf{N}$ *as* $(\mathsf{epk}_i, \mathsf{esk}_i) \leftarrow \mathsf{PK.KeyGen}(1^\lambda)$
– *return* $\mathcal{F}^{resh}_{t,\mathbf{epk}^{t+1}}$ *(defined in Figure 9)*

Theorem 6. *(Security) Let* $\mathcal{D}'(z)$ *be the distribution as in Definition 25. If* TFHE *satisfies semantic security and simulation security (see Def. 10) and satisfies security in presence of key-dependent messages , and* HS *satisfies unforgeability and context-hiding (see Def. 9), then the construction in Fig.8 satisfies security (Def.24) with respect to distribution* $\mathcal{D}'$.

Proof. (Sketch) We point out that most of our proof follows the steps for the UT security in [9]. The main difference is the presence of the $\mathsf{TrapdEval}$ for sUT security. We discuss this later in the proof.

To prove security, we construct a simulator $\mathcal{S} = (\mathcal{S}_S, \mathcal{S}_E)$ for $\mathsf{Expt}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$. The simulator $\mathcal{S}$ is constructed as follows:

– $\mathcal{S}_S(1^\lambda, \mathsf{d}, \mathsf{N})$: This is the simulator for $\mathsf{sUT.Setup}$: it runs the TFHE setup simulator $\mathsf{TFHE}.\mathcal{S}_S$ instead of $\mathsf{TFHE.KeyGen}$ and defines ciphertexts ct_x as an encryption of 0 and $\mathsf{ct}_{\rm tpd}$ as an encryption of $(0, \mathsf{sk}_{\mathsf{hs}}, \rho)$ instead of $(\mathsf{sk}_{\mathsf{fhe}}, \mathsf{sk}_{\mathsf{hs}}, \rho)$.
– $\mathcal{S}_E\left(\mathsf{trapd}, C, C(x)\right)$: it simulates queries of the type (xquery, C). To do that it runs the partial decryption simulator of the TFHE instead of the actual partial decryptor to output fake partial decryptions p_i-s. It then runs the context-hiding simulator of HS passing as input the partial decryptions p_i-s and the function $\mathsf{PartDec}$.

Our adversary has access to two oracles, that for partial evaluations (simulated or not) and those for trapdoor evaluations. We now define a series of hybrids. The change of hybrids $H_{-1} \to H_0$ affects mainly the trapdoor evaluation oracle and it is a different step in the proof of sUT in contrast to that of UT in [9].

The indistinguishability between real and ideal worlds can be argued via a sequence of hybrid experiments between the adversary $\mathcal{A}$ and the challenger.

– H_{-1} is the real security experiment $\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$.
– H_0 is the same as H_{-1} but it produces $\mathsf{ct}_{\rm tpd}$ as an encryption of $(0, \mathsf{sk}_{\mathsf{hs}}, \rho)$ at setup time. This affects the trapdoor evaluation oracle (for queries $(\mathsf{trapdquery}, z)$). This experiment is indistinguishable from $H{-}1$ become of KDM-security of the TFHE scheme (formalized in Section A.3), IND-CPA of PK and the privacy property of Shamir secret sharing: the adversary can open at most $t-1$ ciphertexts thus revealing at most $t-1$ shares of 0 (which will look as if they were from $\mathsf{sk}_{\mathsf{fhe}}$). Notice that we also use the PRF security since it produces the randomness we plug in the sharing algorithm.

– H_1 is the same as H_0, except the challenger simulates the signatures in the $\mathsf{sUT.PartEval}$ By the context-hiding property of HS, the hybrid experiments H_0 and H_1 are computationally indistinguishable.
– H_2 is the same as H_1, except the challenger simulates the partial decryptions by running the TFHE simulator.
By the simulation security of TFHE, the hybrid experiments H_1 and H_2 are statistically indistinguishable.
– H_3 is the same as H_2, except the challenger simulates the setup output pp by running the TFHE simulator. Specifically, the challenger encrypts 0 instead of x and trapd in the setup.
By the semantic security of TFHE, the hybrid experiments H_2 and H_3 are computationally indistinguishable. Also, note that the challenger in H_3 simulates the setup algorithm $\mathsf{sUT.Setup}$ and partial evaluation algorithm $\mathsf{sUT.PartEval}$ without requiring access to the secret x. Hence, H_r corresponds to the ideal experiment $\mathsf{Expt}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{sUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ as simulated through $\mathcal{S}_S$ and $\mathcal{S}_E$ defined above.

D Details on Proactive UT (pUT)

We define a new primitive pUT which can be seen as a proactive version of UT with a set of secret-holders that can change throughout the protocol. We assume that the run of the protocol consists of time periods (epochs) with a handover stage at the beginning of each. At the beginning of each handover stage the challenger produces a set of N ephemeral public-keys. The owners of the corresponding secret keys are the committee members for the new epoch. The output of the resharing stage should be N ciphertexts, each of them containing a pair $(\mathsf{sk}'_i, \sigma'_i)$. The N ciphertexts are not directly an output of the reshare stage but they are "partially evaluated" by the committee members of epoch t. These can be then combined as done for partial evaluations in UT. We let the adversary select $\mathsf{d}-1$ of these shares at each epoch. For the rest the security experiment proceeds as for UT. As we mentioned the set of parties at period t (called committee $\mathcal{C}_t$) can do some partial evaluations and later reshare their shares to the next committee $\mathcal{C}_{t+1}$. Since the communication in each time period should be non-interactive, we require the resharing phase to be publicly verifiable such that everyone, not just the receivers, can verify the correctness of shares.

D.1 Model

We assume a pUT is associated with a public-key encryption scheme PK used to sample the ephemeral keys. In the following we denote by d the threshold for reconstruction and with T the epoch the model is currently referring to. Let $\mathcal{C}_T = \{P_1^{(t)}, \ldots, P_{\mathsf{N}}^{(t)}\}$ be the committee at the time period T. To simplify the notation, we assume that each party $P_i^{(t)}$ (for $i \in [\mathsf{N}]$ and $T = 0, 1, \ldots$) owns an anonymous key pair $(\mathsf{sk}_i^{(t)}, \mathsf{pk}_i^{(t)})$ such that $P_i^{(t)}$ can use $\mathsf{sk}_i^{(t)}$ to decode the encoded shares $[\mathsf{s}^T_{:,i}]$ (encoded by $\mathsf{pk}_i^{(t)}$). Also, we use the notation $[\mathbf{s}_\mathbf{j}^\mathbf{T}]$ in the

$\mathsf{VfyReshare}$ algorithm to denote the set of encoded shares $P_i^{(t)}$ has received from the previous committee, namely $\{[\mathsf{s}^{(t)}_{i_1,j}],\ldots,[\mathsf{s}^{(t)}_{i_\mathsf{d},j}]\}$.

A proactive universal thresholdizer scheme for $\mathbb{S}$ consists of a tuple of PPT algorithms $\mathsf{pUT} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{VfyEval}, \mathsf{Combine}, \mathsf{Reshare}, \mathsf{VfyReshare}, \mathsf{Reconstruct})$ such that $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{VfyEval}, \mathsf{Combine}$ are defined similarly to the UT algorithms and the reshare algorithms are defined as in 6.1.

The evaluation correctness and robustness properties are similar to the ones for UT schemes, except in pUT, we require two additional correctness properties *Resharing Correctness* and *Share Reconstruction Correctness*. Informally, *Resharing Correctness* states that proper computation of $\mathsf{pUT.Reshare}$ verifies by the $\mathsf{pUT.VfyReshare}$ algorithm. *Share Reconstruction Correctness* states that for d valid shares (computed by $\mathsf{pUT.Reshare}$), the reconstruction algorithm $\mathsf{pUT.Reconstruct}$ computes a valid share. We also require an additional *robustness property for resharing* requiring that no adversary corrupting at most $\mathsf{d}-1$ parties can compute the correct share value.

Definition 26 (pUT Security). *A pUT scheme satisfies security with respect to distribution D if for all λ, the following holds. There exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_S, \mathcal{S}_E)$ such that for any PPT adversary $\mathcal{A}$, the following experiments $\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{pUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ and $\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{pUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$ are computationally indistinguishable.*

$\underline{\mathsf{Expt}^{\mathsf{world}\in\{\mathsf{real},\mathsf{ideal}\}}_{\mathcal{A},\mathsf{pUT}}\left(1^\lambda, \mathsf{d}, \mathsf{N}\right):}$

```
1. $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{d}, \mathsf{N})$
2. if $\mathsf{world} = \mathsf{real}$ then $(\mathsf{pp}, \mathsf{s}_{1,0}, \ldots, \mathsf{s}_{\mathsf{N},0}, \mathsf{trapd}) \leftarrow \mathsf{pUT.Setup}\left(1^\lambda, \mathsf{d}, \mathsf{N}, x\right)$
3. else if $\mathsf{world} = \mathsf{ideal}$ then $(\mathsf{pp}, \mathsf{s}_{1,0}, \ldots, \mathsf{s}_{\mathsf{N},0}, \mathsf{trapd}) \leftarrow \mathcal{S}_S\left(1^\lambda, \mathsf{d}, \mathsf{N}\right)$.
4. $\mathcal{A}$ outputs a corruption set $\mathcal{C}_0$ of size $\mathsf{d}-1$
5. The challenger provides the shares $\{\mathsf{s}_{i,0}\}_{i\in\mathcal{C}_T}$ to $\mathcal{A}$.
6. For $T = 1, \ldots, \mathsf{poly}(\lambda)$:
   (a) $\mathcal{A}$ outputs a corruption set $\mathcal{C}_T$ of size $\mathsf{d}-1$
   (b) $\mathcal{A}$ can ask for a polynomial number of adaptive queries to the oracle
       $\mathcal{O}^{\mathsf{world}}_{\mathsf{pUT}}$ (defined below).
   (c) The challenger samples ephemeral key $(\mathsf{epk}_i, \mathsf{esk}_i)_{i\in[\mathsf{N}]} \leftarrow$
       $\mathsf{PK.KeyGen}(1^\lambda)^\mathsf{N}$, calls $\mathsf{pUT.Reshare}(\mathsf{pp}, \mathsf{s}_{i,T}, \mathbf{epk})$ for all $i \in [\mathsf{N}]$
       and returns the outputs to $\mathcal{A}$ together with the $\mathsf{epk}_i$-s.
   (d) The challenger uses $\mathsf{pUT.Reconstruct}$ with input each $\mathsf{esk}_i$ to reconstruct
       new shares $\{\mathsf{s}_{i,T+1}\}_{i\in[\mathsf{N}]}$.
   (e) The challenger provides the shares $\{\mathsf{s}_{i,T}\}_{i\in\mathcal{C}_T}$ to $\mathcal{A}$.
7. Adversary outputs a guess $\mathsf{guess}$
8. Return 1 iff $\mathsf{guess} = \mathsf{world}$
```

We now define oracle $\mathcal{O}^{\mathsf{world}}_{\mathsf{pUT}}$. It can receive as a query C where C is a circuit and the oracle responds with:

– if $\mathsf{world} = \mathsf{real}$ then returns $\{\mathsf{y}_i \leftarrow \mathsf{pUT.Eval}(\mathsf{pp}, \mathsf{s}_{i,T}, C)\}_{i\in[\mathsf{N}]}$
– if $\mathsf{world} = \mathsf{ideal}$ then return $\{\mathsf{y}_i\}_{i\in[\mathsf{N}]} \leftarrow \mathcal{S}_E(\mathsf{trapd}, C, C(x))$

D.2 Construction

Our construction is in Figure 9. It uses a secure sUT as the underlying mechanism and almost uses it completely black-box, nonetheless it relies only the specific underlying structure of the TFHE as described in the sUT section (the fact that its partial secrets are a resharing of a specific secret $\mathsf{sk}_{\mathsf{fhe}}$. We leave as future work how to generalize these results. To ensure robustness one needs to slightly modify the sUT construction so that shares are signed with the tag "(i, t)" instead of only "i". We can do this easily by extending the syntax of the setup for sUT with an additional auxiliary parameter, a session id that can represent epoch or any other information.

We now prove the security property (Definition 26). Other properties are straightforwardly derived as for sUT. It is important to notice that: for correctness we rely on the special structure of the TFHE key as shares of a secret; for the robustness property of resharing we rely on security of the PRF and unforgeability of the homomorphic signature (the intuition is that if an adversary will only gain $\mathsf{d}-1$ signatures of the type "(i, t)" for each epoch t and will not be able to provide a d-th one without forging).

Theorem 7. *(Security) Consider the sUT construction in Figure 8 then the construction in Figure 9 satisfies security for pUTs (Definition 26).*

Proof. (Sketch) We rely on the fact that the sUT construction is secure w.r.t. distribution D' (Definition 25). We claim that if an adversary $\mathcal{A}_p$ is able to distinguish the ideal and real worlds in the pUT experiment, then we are able to construct a successful adversary $\mathcal{A}_s$ for the sUT experiment. The adversary $\mathcal{A}_s$ would work as follows:

– The adversary $\mathcal{A}_s$ forwards the sUT public parameters to $\mathcal{A}_p$ and selects the same corruption set.
– For each query to the partial evaluation oracle from $\mathcal{A}_p$, adversary $\mathcal{A}_s$ queries its own partial evaluation oracle with the same input and returns the result.
– To simulate a resharing step from epoch t to epoch $t+1$, $\mathcal{A}_s$ first samples $\mathsf{d}-1$ fresh ephemeral key pairs $(\hat{\mathsf{epk}}_i, \hat{\mathsf{esk}}_i)_{i\in[\mathsf{d}-1]}$[12]. It then invokes its own trapdoor evaluation algorithm with input $(t, (\mathsf{epk}_i)_{i\in[\mathsf{d}-1]})$. It then receives a vector $\mathbf{y}^{\mathrm{resh}}$ of N shares of $\mathbf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_{\mathsf{N}})$ the output of $\mathcal{F}^{\mathrm{resh}}_{t,\mathbf{epk}^{t+1}}$. It will also receive the remaining epk_i-s (returned as C by the trapdoor evaluation oracle). This vector is $\mathbf{y}^{\mathrm{resh}}$ forwarded to $\mathcal{A}_p$ together with the ephemeral public keys as if it were step (c) of the pUT security experiment.

[12] For simplicity we express this with indices $1, \ldots, \mathsf{d}-1$ but they should correspond to the indices declared in the corruption set by $\mathcal{A}_p$.

- In order to distribute the shares for the next round (step (d)), the $\mathcal{A}_s$ uses $\mathsf{sUT.Combine}$ on $\mathbf{y}^{\text{resh}}$ and retrieves the N ciphertexts output of $\mathcal{F}^{\text{resh}}_{t,\mathbf{epk}^t}$. It then uses the $\mathsf{d}-1$ keys $\hat{\mathsf{esk}}_i$-s to open the related ciphertexts returns to $\mathcal{A}_s$ $\mathsf{d}-1$ reconstructed shares, $\mathcal{A}_s$ forwards them to $\mathcal{A}_p$ together with $\mathbf{y}^{\text{resh}}$.
- At the end of the protocol $\mathcal{A}_s$ outputs the same guess as $\mathcal{A}_p$

We observe that $\mathcal{A}_s$ simulates the view of $\mathcal{A}_p$ in the pUT security experiments. The advantage of $\mathcal{A}_s$ is thus negligibly close to that of $\mathcal{A}_p$. Invoking security of sUT concludes the proof.