Received 23 January 2024, accepted 10 February 2024, date of publication 16 February 2024, date of current version 23 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3366688

RESEARCH ARTICLE

A Dynamic Proactive Secret Sharing Scheme for Quadratic Functions

HONG CHEN AND LIANGFENG ZHANG
School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China
Corresponding author: Liangfeng Zhang (zhanglf@shanghaitech.edu.cn)

ABSTRACT *Dynamic proactive secret sharing* (DPSS) allows a client to share a secret s among a committee of n participants and allows the committee to reshare the secret among a new committee of n participants such that the secret s can be reconstructed by any authorized subset of the new committee but an adversary that corrupts both members from the old committee and members from the new committee learns no information about s. The existing DPSS schemes only allow the old committee to reshare the *same* secret s to the new committee. In this paper, we consider *dynamic proactive functional secret sharing (DPFSS)*, a new model for DPSS that allows the old committee to reshare functions of the original secret s among the new committee, and construct a DPFSS scheme for resharing the evaluation results of *quadratic* functions on the original secret. We discuss the applications of our DPFSS scheme in distributed systems such as those for health data sharing and smart meter billing. Our scheme is efficient and achieves a communication complexity of $\mathcal{O}(n^2)$ and a time complexity of $\mathcal{O}(n)$. Our experimental results show that for a committee of size $n = 100$, the communication cost is ≈1.166 MB and the time cost is ≈10.72 s.

INDEX TERMS Secret sharing, dynamic committee, function handoff, proactive security.

I. INTRODUCTION

Secret sharing (SS) allows a client to share a secret s among a group of participants $\mathcal{C} = \{P_1, \ldots, P_n\}$ by distributing a share of the secret to every participant such that any authorized subset of $\mathcal{C}$ can recover s from their shares but any unauthorized subset of $\mathcal{C}$ learn no information about s. The family Γ of all the authorized subsets is called an access structure and satisfy the monotone property: if $A \in \Gamma$ and $A \subseteq B$, then $B \in \Gamma$. Shamir [1] and Blakley [2] introduced the notion of secret sharing and constructed the first SS schemes. The access structure Γ of their schemes consists of all subsets of $\mathcal{C}$ of cardinality $> t$, i.e., $\Gamma = \{A \subseteq \mathcal{C} : |A| > t\}$, and the resulting schemes have been called threshold SS schemes (TSSS). Ito et al. [3] considered arbitrary access structures that satisfy the monotone property and proposed several general constructions. Since [1], [2], [3], secret sharing has received widespread attention and found numerous applications in the design of many cryptographic protocols such as secure multiparty computations (MPC) [4], [5], private information retrieval (PIR) [6], threshold cryptosystems [7], identity-based encryption [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

Dynamic proactive secret sharing schemes allow the group of participants to dynamically change and achieve the following security guarantee: even if a limited number of participants from both the old group and the new group collude with each other, they still learn no information about the shared secret. Both the properties of "proactive" and "dynamic" have a long history. Ostrovsky and Yung [9] considered a *proactive* security model for secret sharing where the lifetime of a secret s is divided into phases and the shares are periodically refreshed at the beginning of every phase. In this model, there may be a *mobile* adversary that corrupts a number of parties in every phase, accumulate the shares of the corrupted parties and then compute information of the secret s. Schemes in this model have been called *proactive secret sharing* (PSS) schemes. The security of PSS schemes requires that the mobile adversary cannot learn any information about the secret s, provided that the number of corrupted parties in each phase is below a threshold. There are many standard techniques for constructing PSS schemes.

For example, Herzeberg et al. [10] uses a set of random shares of 0 to update the shares of s at the beginning of every phase and obtain a PSS scheme for the threshold access structure. Frankel et al. [11] utilized a similar resharing protocol to update the old shares held by the parties.

In most existing PSS schemes [10], [11], [12], both the set of participants and the access structure are never changed. However, in many real-life scenarios [13], [14], [15], the participants may join and leave a data management system arbitrarily and thus result in dynamic groups of participants. *Dynamic proactive secret sharing* (DPSS) [16], [17], [18] allows a client to share a secret s among a committee $\mathcal{C} = \{P_1, \ldots, P_n\}$ of n participants such that $\mathcal{C}$ is able to pass s to a new committee $\tilde{C} = \{\tilde{P}_1, \ldots, \tilde{P}_n\}$ in the form of shares, where the new committee may differ from $\mathcal{C}$. A DPSS scheme may consist of three phases: *share generation, handoff* and *reconstruction*. In the share generation phase, a client shares a secret s among a committee $\mathcal{C}$ by distributing a share of s to every participant in $\mathcal{C}$. In the handoff phase, participants in $\mathcal{C}$ may leave the committee and new participants may join so that the old committee $\mathcal{C}$ turns into a new committee $\tilde{\mathcal{C}}$. Then the shares of s held by the participants in $\mathcal{C}$ are updated and the updated shares are sent to the participants in $\tilde{\mathcal{C}}$. In the reconstruction phase, the secret s can be recovered by any authorized subset of the participants in $\tilde{\mathcal{C}}$ with their shares. In DPSS schemes, we usually consider a mobile and active adversary $\mathcal{A}$ that can corrupt $\leq t$ participants from $\mathcal{C}$ and $\leq t$ participants from $\tilde{\mathcal{C}}$. Such schemes satisfy the property of *integrity* if s can be correctly reconstructed in the presence of $\mathcal{A}$ and satisfy the property of *secrecy* if the adversary $\mathcal{A}$ learns no information about s.

Desmedt and Jajodia [16] introduced the notion of DPSS and constructed a DPSS scheme, which may be used to store secret data among dynamic committees. Since [16], DPSS has gained great popularity in long-term secure storage and various DPSS constructions have been proposed. The existing DPSS schemes are constructed either under a synchronous network or under an asynchronous network. The efficiency of such schemes is mainly measured by their *communication complexity*. Under a *synchronous* network, Wong et al. [19] constructed the first verifiable DPSS scheme, which enables the participants to verify the correctness of protocol executions by using Feldman's polynomial commitment [20]. Their scheme satisfies the properties of integrity and secrecy in the presence of an adversary that corrupts $\leq t$ participants from $\mathcal{C}$ and $\leq t$ participants from $\tilde{\mathcal{C}}$. Their communication complexity is exponential in the number $n = |\mathcal{C}|$ and thus makes the scheme impractical. Baron et al. [21] proposed a batched DPSS scheme, which enables one to store a batch of secrets at a time. The scheme utilizes the Berlekamp-Welch algorithm [22] to guarantee the correctness of reconstruction. It achieves a total communication complexity of $\mathcal{O}(n^3)$ for $n = |\mathcal{C}|$ and the optimal amortized communication complexity of $\mathcal{O}(1)$. As a drawback, their scheme must reconstruct all secrets at one shot and cannot recover any single secret separately. Goyal et al. [17] constructed a batched DPSS scheme, where each secret can be individually recovered. They apply KZG commitment [23] to verify the correctness of the shares in their protocol executions and achieve the optimal total communication complexity of $\mathcal{O}(n^2)$ for $n = |\mathcal{C}|$.

With a time bound for successful message deliveries, some deviations of the corrupted parties in a DPSS scheme can be identified efficiently under synchronous networks. For asynchronous networks, the DPSS scheme executions often rely on a challenge-response mechanism [24], [25] to propose and decide on the validity of the messages, where one or more participants propose a message to all participants and a same decision on the message will eventually be output by the majority. In particular, a sender may keep retransmitting a message such that it will eventually be delivered with a maximum delay bounded by an unknown variable and a receiver may respond to confirm for having received the messages from some certain senders. As a result, the asynchronous networks may incur a large communication overhead, compared with the synchronous networks. Zhou et al. [26] constructed the first DPSS scheme under an asynchronous network. Their scheme can tolerate Byzantine faults that may cause the participants to behave arbitrarily badly (e.g. expose messages), but has an exponential communication complexity $\exp(n)$ for $n = |\mathcal{C}|$. Subsequently, the communication complexity of DPSS under asynchronous networks has be reduced to $\mathcal{O}(n^4)$ by Schultz et al. [25] and to $\mathcal{O}(n^3)$ by Hu et al. [27] and Yurek et al. [28], respectively. The schemes of [27] and [28] are known as most efficient to date.

The existing DPSS schemes have been focusing on securely passing *the same secret* s between dynamic committees and minimizing the communication complexity. In many real-life scenarios, instead of the secret s itself, one may expect to pass *functions* of the secret s between different committees. For examples, in glaucoma care [29], a user of the smart soft contact lenses would like to learn an average of the continuous 24-hour measurements of intraocular pressure rather than obtains piles of real-time data. In [30], secure sharing of personal health data, which includes highly sensitive physiological information, among healthcare institutions is quite important. One may need to restrict the information that can be accessed by the institutions with different trust level. The institutions with higher trust levels may be allowed to achieve the information, which has a strong correlation with the original data. But the institutions with lower trust levels are restricted to do some fuzzy analyses on the data.

To our knowledge, no existing DPSS schemes allow one to transfer the *evaluation results* of more general functions on the shared secret data between different committees. Such schemes may help extract *useful features* from the secret data, without revealing unnecessary information about the data to the participants/users. Therefore, given the application scenarios, it is interesting to construct DPSS schemes that allow an old committee $\mathcal{C}$ to pass functions of the secrets shared among the members of $\mathcal{C}$ to a new committee $\tilde{\mathcal{C}}$.

A. OUR CONTRIBUTIONS

In this paper, we propose *dynamic proactive functional secret sharing (DPFSS)*, a new model of DPSS that transfers *functions* of the shared secrets between *different* committees. Compared with the existing DPSS, the new and more general functionality of passing functions of the shared secrets from an old committee to a new committee serves as the main novelty of our DPFSS model. We construct a scheme in this new model and show its applications.

Our Model: Our DPFSS model consists of three phases called *share generation, function handoff* and *reconstruction*. At a high-level, in the share generation phase, an input client shares a set $\mathbf{s}$ of secrets among a committee $\mathcal{C} = \{P_1, \dots, P_n\}$ of n parties. In the function handoff phase, N functions $\{F_k\}_{k=1}^{N}$ of $\mathbf{s}$ are transferred from the old committee $\mathcal{C}$ to a new committee $\tilde{\mathcal{C}} = \{\tilde{P}_1, \dots, \tilde{P}_n\}$. In the reconstruction phase, an output client reconstructs $\{F_k(\mathbf{s})\}_{k=1}^{N}$. A DPFSS scheme is said to be *secure* if no active and mobile PPT adversary with a corruption threshold $t(< n/3)$ can fool an honest output client to output a value $\neq F_k(\mathbf{s})$, $k \in [N]$ (*integrity*) or learn any information about $\mathbf{s}$ or $\{F_k(\mathbf{s})\}_{k=1}^{N}$ other than the public information (*secrecy*).

Our Construction: We construct a scheme DPFSS for transferring *quadratic* functions of the shared secret data between different committees. The scheme consists of three algorithms or protocols $\mathsf{DPFSS.S}$, $\mathsf{DPFSS.FH}$ and $\mathsf{DPFSS.R}$ that respectively realize the three phases in our DPFSS model.

In the share generation phase ($\mathsf{DPFSS.S}$), the input client uses *Shamir's secret sharing scheme* [1] to generate a *sharing* (a vector of shares) of every secret in $\mathbf{s}$ and distributes the shares among the old committee $\mathcal{C}$. The sharings of $\mathbf{s}$ are committed by the *Feldman's polynomial commitment scheme* [20] such that the correctness of every share in one sharing can be verified. At the end of this phase, every party in $\mathcal{C}$ holds a share of each secret in $\mathbf{s}$.

The function handoff phase ($\mathsf{DPFSS.FH}$) includes two sub-phases: prepare and refresh. In the prepare phase, for all $k \in [N]$, participants in both the old committee $\mathcal{C}$ and the new committee $\tilde{\mathcal{C}}$ jointly compute a *random coupled sharing* (RCS), which consists of two sharings of the *same* value $r_k (= \tilde{r}_k)$, for transferring the evaluation result $F_k(\mathbf{s})$. The $2N$ sharings are committed with Feldman's polynomial commitments as well. In the refresh phase, for all $k \in [N]$ parties in $\mathcal{C}$ jointly pass a value $F_k(\mathbf{s}) + r_k$ to the parties in $\tilde{\mathcal{C}}$. With the sharing of $\tilde{r}_k$, each party in $\tilde{\mathcal{C}}$ can compute a share in the sharing of $F_k(\mathbf{s})$. During the executions, the shares are verified with the commitments.

In the reconstruction phase ($\mathsf{DPFSS.R}$), every party in $\tilde{\mathcal{C}}$ sends its shares to the output client. By applying the Shamir's secret sharing, the output client outputs N evaluation results $\{F_k(\mathbf{s})\}_{k=1}^{N}$.

B. APPLICATIONS

Our DPFSS scheme may be applied in many scenarios that require privacy-preserving storage, dynamic membership and verifiable computation of linear or quadratic functions.

Health Data Sharing: Health data management systems [30], [31], [32] allow a client to maintain personal health data and share the data with healthcare institutions. For example, the health data sharing system of [32] may consist of a client, a cloud database, healthcare institutions, and a blockchain network. In such a system, the wearable devices of the client collect personal health data of the client, upload the data to the cloud database, and record both the hash value of the data and a set of access policies on the blockchain. The blockchain plays the role of a platform that makes both the hash value and the access policies public and tamper resistant. Upon receiving a healthcare institution's request of accessing the data, the cloud database may check the access policy on the blockchain and grant access if that institution is really authorized. Upon receiving the data, the healthcare institution may check its integrity with the hash value on the blockchain. Health data such as blood pressure and electrocardiogram (ECG) reports may include highly sensitive physiological information and make secure sharing a crucial requirement. While systems such as [32] give a solution with access control, the cloud database is a single point of failure and requires expensive maintainance. Goyal et al. [17] proposed extractable witnesse encryption on blockchain (eWEB), which is based on DPSS and may avoid single of point failure using a decentralized system. Using eWEB, the client's wearable devices may record the access policy (a release condition) on chain and share the personal health data off chain among a cluster of nodes in the blockchain network. If the healthcare institution has a witness (i.e., permission) to the release condition, it may send a proof of the witness to the cluster and publish the hash value of the proof on chain. The cluster may check the validity of both the hash value and the proof. If they are valid, the healthcare institution is granted access. Such a system allows nodes to leave and join dynamically, and thus requires a DPSS to support membership changes. Whenever the institutions are only interested in the results of certain analysis on the data, the DPSS-based system of [17] may leak too much information to the institutions. By applying our DPFSS (See Fig. 1), the old committee of nodes can directly transfer the results of requested analysis to a new committee of nodes such that the institutions are only able to learn the expected results rather than the original health data.

Smart Meter Billing: The smart meter [33], [34] allows one to transmit a set of data points that record residential energy consumption for short time intervals (e.g. every 15 minutes). Then the consumption will be billed with dynamic, time-of-use tariffs. However, the data may expose the behavioral

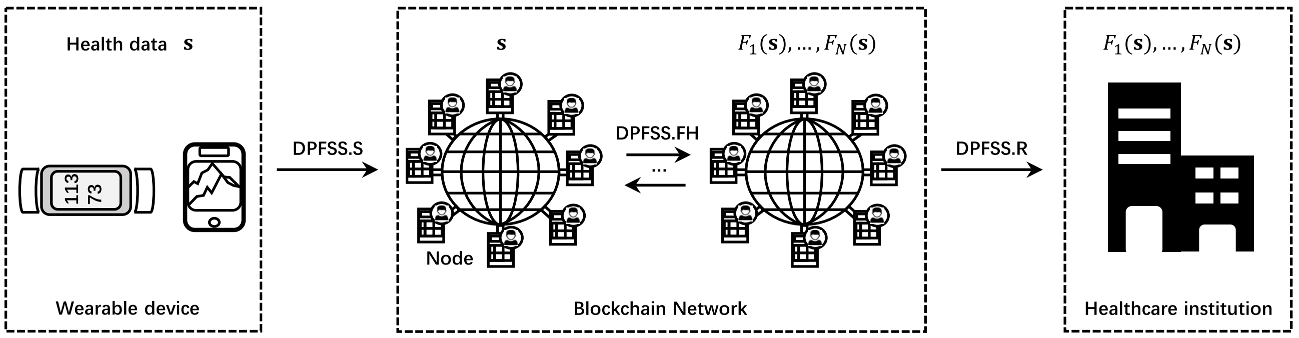


FIGURE 1. **DPFSS-based health data sharing.**

patterns of residents. Our DPFSS scheme can provide a privacy-preserving storage of the consumption data collected by smart meters and conduct a verifiable computation of the bill. In particular, by applying DPFSS in the time-of-use billing scheme [33], the consumption profile $\mathbf{c} = (c_1, \ldots, c_n)$ uploaded by a smart meter, where every c_i represents the amount of utility used in the interval i of one day, can be safely stored among a set of servers. Then with the time-of-use tariff $\mathbf{v} = (v_1, \ldots, v_n)$, where v_i prices every interval i, DPFSS enables the servers to jointly calculate and output the time-of-use price $P(\mathbf{c}, \mathbf{v}) = \sum_{i=1}^{n} c_i v_i$ of the consumption of one day, without leaking additional information about $\mathbf{c}$.

C. RELATED WORK

There are some existing secret sharing schemes devised for computing functions. However, such schemes neither support the *dynamics* of the access structure nor satisfy *proactive security* in a long-lived system.

Verifiable Secret Sharing (VSS): VSS schemes consider an active adversary that may fully control the behaviors of the participants or even the dealer to prevent a consistent reconstruction. Such schemes [35], [36], [37] allow the honest participants to reconstruct the same secret in the presence of an active adversary. Constructions of VSS schemes usually apply a commitment scheme to check the correctness of shares. For example, Feldman [20] proposed a commitment scheme which commits a polynomial $f(X) = a_0 + a_1 X + \cdots + a_t X^t$ with $\mathsf{Com} = (g^{a_0}, \ldots, g^{a_t})$, where g is a random generator of a cyclic group. Therefore, the correctness of an evaluation (x, y) can be checked with the equality $g^y = \prod_{i=0}^{t} (g^{a_i}))^{x^i}$. Their scheme is computational hiding, under the DLOG assumption. Kate et al. [23] proposed two unconditional hiding commitment schemes where the commitment reveals no information about the polynomial. In VSS schemes, the shares held by the participants are *never refreshed* and the committee of participants is not dynamic but *static*.

Multiplicative Secret Sharing (MSS): A m-MSS enables a client to share m $(m > 2)$ secrets $s_1, \ldots, s_m$ among a group of parties such that $s_1 \times \cdots \times s_m$ can be reconstructed without recovering any single secret. Ben-Or et al. [38] proposed the first MSS scheme for computing the multiplication of two secrets in the presence of a passive adversary. By applying the Berlekamp Welch algorithm, Damgård and Nielsen [39] constructed an unconditionally secure MSS scheme that can correct a threshold number of errors. Yoshida and Obana [40] introduced verifiable m-MSS and proposed a scheme [41] for multiplicative-only homomorphism. MSS schemes have *static* access structures and do not consider proactive security.

Homomorphic Secret Sharing (HSS): HSS allows a client to share a secret s among servers and every server in an authorized subset can compute a function f on its share to obtain a partial result such that these partial results allow the reconstruction of $f(s)$. Boyle et al. [42] introduced HSS and constructed a 2-server HSS for evaluating any branching programs. Verifiable HSS was studied in [43] and [44], where the correctness of the reconstructed value is checked with verification techniques. The access structures of HSS schemes are *static* and the loss of partial results (e.g., caused by latency or down of servers) may cause a failure to recover $f(s)$.

Function Secret Sharing (FSS): FSS allows a client to *split a function* f into n function shares $\{f_i\}_{i=1}^{n}$, each of which perfectly hides f, such that $f(s)$ can be recovered from the n partial results $f_1(s), \ldots, f_n(s)$ for any s. Boyle et al. [45] introduced FSS and constructed a verifiable FSS scheme in [46]. Luo et al. [47] proposed an information-theoretic secure (t, n)-threshold FSS, where any $(t-1)$ function shares unconditionally hide f. In FSS schemes, the input s is *public* to all participants and the function f is secret from any participant.

D. ORGANIZATION

In Section II, definitions and techniques related to our DPFSS constructions are given. In Section III, our DPFSS model is defined. In Section IV, we construct a DPFSS scheme and formally prove its security. In Section V, we implement our construction and evaluate its performance. Section VI shows the conclusions of our work.

II. PRELIMINARIES

Let κ be a security parameter. We say that a function $\epsilon(\kappa)$ is *negligible* in κ and denote $\epsilon(\kappa) = \mathsf{negl}(\kappa)$ if for any $c > 0$, there is an integer κ_c such that $\epsilon(\kappa) < \kappa^{-c}$ for all $\kappa > \kappa_c$. We say that a function $f(\kappa)$ is *polynomial* in κ and denote $f(\kappa) = \mathsf{poly}(\kappa)$ if there exists $c > 0$ such that $f(\kappa) = O(\kappa^c)$. In this paper, we consider *probabilistic polynomial time* (PPT) algorithms that run in time $\mathsf{poly}(\kappa)$. For any integer $n > 0$, we denote $[n] = \{1, \ldots, n\}$. For any prime p, we denote by $\mathbb{F}_p$ the finite field of p elements and denote by $\mathbb{F}_p[X]$ the ring of univariate polynomials with coefficients from $\mathbb{F}_p$. We will use bold letters to denote vectors and matrices. Let $\mathbf{a}, \mathbf{b}$ be vectors of length n. For any $i \in [n]$, we denote by $\mathbf{a}[i]$ the i-th element of $\mathbf{a}$. We denote $\mathbf{a} \circ \mathbf{b} = (a[1] \cdot b[1], \ldots, a[n] \cdot b[n])$, $\mathbf{a} \oslash \mathbf{b} = (a[1]/b[1], \ldots, a[n]/b[n])$ and $a^c = ((a[1])^c, \ldots, (a[n])^c)$. For any matrix $\mathbf{M}$, we denote by $\mathbf{M}_{i,j}$ its (i, j)-entry. We denote by $\mathbf{1}_n = (1, \ldots, 1)$ the all-one vector of length n. For any finite set A, we denote by '$a \leftarrow A$' the process of choosing an element a uniformly from A.

Discrete Logarithm (DLOG) Assumption: Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order $p \approx 2^\kappa$. We say that the *DLOG assumption* is true in $\mathbb{G}$ if for any PPT adversary $\mathcal{A}$, $\Pr[\mathcal{A}(g, g^c) = c] \leq \epsilon(\kappa)$, where the probability is taken over $c \leftarrow \mathbb{F}_p$ and the randomness of $\mathcal{A}$.

Bilinear Map: In this work, we use a bilinear group generator [48] $\mathcal{BG}(\kappa)$ that takes a security parameter κ as input and outputs a *bilinear group context* $\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ are cyclic groups of prime order $p \approx 2^\kappa$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a map with the following properties:

- *bilinear*: $e(g^a, g^b) = e(g, g)^{ab}$ for any $a, b \in \mathbb{F}_p$;
- *non-degenerate*: $e(g, g)$ is a generator of $\mathbb{G}_T$;
- *efficiently computable*: There is a polynomial-time algorithm computing e.

A. SHAMIR'S SECRET SHARING

For any integers t, n such that $n \geq t > 0$, Shamir's $(t + 1, n)$*-threshold secret sharing scheme* $\mathsf{SSS} = (\mathsf{SSS.Share}, \mathsf{SSS.Recon})$ [1] allows a *dealer* to share a secret s among n *participants* $P_1, \ldots, P_n$ such that: (i) any $\geq t+1$ participants can recover s with their shares; (ii) any $\leq t$ participants learn no information about s. Formally, it consists of two algorithms as follows:

- $[s]_t \leftarrow \mathsf{SSS.Share}(t, n, s)$. The share generation algorithm shares a secret $s \in \mathbb{F}_p$ among the n participants by choosing a polynomial

$$f(X) = s + \sum_{i=1}^{t} a_i X^i \tag{1}$$

  with random coefficients $a_1, \ldots, a_t \leftarrow \mathbb{F}_p$ and sending to every participant P_i a share $s_i = f(i)$. In particular, the vector $[s]_t = (s_1, \ldots, s_n) = (f(1), \ldots, f(n))$ of all shares is called a *sharing* of s under SSS.

- $s \leftarrow \mathsf{SSS.Recon}(\{s_i\}_{i\in I})$. Given a set $\{s_i\}_{i\in I}$ of shares, where $I \subseteq [n]$ is a subset of cardinality $> t$, the reconstruction algorithm recovers s as $s = \sum_{i\in I} s_i \ell_{I,i}$ by doing Lagrange interpolation, where

$$\ell_{I,i} = \prod_{z\in I, z\neq i} z/(z - i) \tag{2}$$

  is a Lagrange coefficient for all $i \in I$.

Random Coupled Sharing (RCS): A random coupled sharing (RCS) $([r]_t, [\tilde{r}]_t)$ consists a pair of sharings $[r]_t = (f_r(1), \ldots, f_r(n))$, $[\tilde{r}]_t = (f_{\tilde{r}(1)}, \ldots, f_{\tilde{r}}(n))$, where $f_r(X)$ and $f_{\tilde{r}}(X)$ are randomly chosen degree-t polynomials subject to $f_r(0) = f_{\tilde{r}}(0)$. In this work, the random coupled sharings will be utilized to transfer quadratic functions of secret data from the old committee to the new committee.

B. POLYNOMIAL COMMITMENT

A polynomial commitment scheme allows one to commit to a polynomial f by publishing a commitment com_f and later open the evaluation of the polynomial f at any input i (i.e., $y = f(i)$) such that a verifier can check the correctness of y with com_f. The scheme is called *hiding* if com_f leaks no information about f and *binding* if the committer cannot open com_f to a value $y' \neq f(i)$ that successfully passes the verification.

In Feldman's work [20], a discrete logarithm based polynomial commitment scheme $\mathsf{FC} = (\mathsf{FC.Setup}, \mathsf{FC.Commit}, \mathsf{FC.Verify})$ has been applied to the sharing polynomial f in Eq. (1) to construct a verifiable secret sharing scheme. In this work, we will use FC to verify the correctness of the shares under SSS as well. Note that the sharing $[s]_t$ in Section II-A may be identified with the sharing polynomial f in Eq. (1). We will interchangeably say FC is a commitment scheme for polynomials or for sharings. For completeness, the scheme FC can be described as follows:

- $\mathsf{ck} \leftarrow \mathsf{FC.Setup}(\kappa)$: Take the security parameter κ as input, choose a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order $p \approx 2^\kappa$, and output a commitment key $\mathsf{ck} = (\mathbb{G}, g)$.
- $\mathsf{com}_f \leftarrow \mathsf{FC.Commit}(\mathsf{ck}, f)$: Take a polynomial $f(X) = s + a_1X + \cdots + a_tX^t$ as input and output a commitment $\mathsf{com}_f = (g^s, g^{a_1}, \ldots, g^{a_t})$.
- $\{0, 1\} \leftarrow \mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_f, i, y)$: To verify the correctness of y as $f(i)$, compute a witness $\eta_i = \prod_{z=0}^{t} \mathsf{com}_f[z + 1]^{i^z} (= g^{f(i)})$. If $g^y = \eta_i$, output 1; otherwise, output 0.

The scheme FC is unconditionally binding and computationally hiding under the DLOG assumption in $\mathbb{G}$. Furthermore, it is *linearly homomorphic*: for any constants $\alpha, \beta \in \mathbb{F}_p$ and any polynomials $f_1(X), f_2(X) \in \mathbb{F}_p[X]$ of degree $\leq t$, say $f_1(X) = s_1 + a_1X + \cdots + a_tX^t$ and $f_2(X) = s_2 + b_1X + \cdots + b_tX^t$,

$$\mathsf{com}_{\alpha f_1+\beta f_2} = (\mathsf{com}_{f_1})^\alpha \circ (\mathsf{com}_{f_2})^\beta. \tag{3}$$

III. DPFSS MODEL

A dynamic proactive secret sharing scheme for quadratic functions involves an *input client* P_{in}, an *output client* P_{out}, an *old committee* $\mathcal{C} = \{P_1, \dots, P_n\}$ and a *new committee* $\tilde{\mathcal{C}} = \{\tilde{P}_1, \dots, \tilde{P}_n\}$, where $\{P_{in}, P_{out}\} \cap (\mathcal{C} \cup \tilde{\mathcal{C}}) = \emptyset$. It allows the input client to share a vector $\mathbf{s} = (s_1, \dots, s_m)$ of secrets among the members of $\mathcal{C}$ such that later $\mathcal{C}$ is able to transfer the evaluations of $N(\leq n-t)$ functions $\{F_k\}_{k=1}^N$ at s to $\tilde{\mathcal{C}}$ and the output client is able to reconstruct the N values $\{F_k(s)\}_{k=1}^N$ with shares from the new committee $\tilde{\mathcal{C}}$.

Definition 1 (Dynamic proactive functional secret sharing): A DPFSS scheme $\mathsf{DPFSS} = (\mathsf{DPFSS.S}, \mathsf{DPFSS.FH}, \mathsf{DPFSS.R})$ consists of three algorithms or protocols that can be described as follows:

- $(\{\mathbf{s}_j\}_{j=1}^n, \mathsf{aux}) \leftarrow \mathsf{DPFSS.S}(\kappa, t, n, \mathbf{s})$. The *share generation* algorithm takes a security parameter κ, a threshold $t(< n/3)$, the size n of committees and a secret vector $\mathbf{s} = (s_1, \dots, s_m)$ as input and outputs both n shares $\{\mathbf{s}_j\}_{j=1}^n$ of s and a piece of public verification information aux. The input client will run this algorithm, distribute a share s_j to every participant P_j in the old committee $\mathcal{C}$ and make aux public.
- $(\{\tilde{\mathbf{s}}_j\}_{j=1}^n, \widetilde{\mathsf{aux}}) \leftarrow \mathsf{DPFSS.FH}(\{F_k\}_{k=1}^N, \{\mathbf{s}_j\}_{j=1}^n, \mathsf{aux})$. The *function handoff* protocol enables the committee $\mathcal{C}$ with inputs $s_1, \dots, s_n$, aux to share the $N(\leq n-t)$ values $\{F_k(s)\}_{k=1}^N$ to $\tilde{\mathcal{C}}$. At the end of this protocol, each party $\tilde{P}_j \in \tilde{\mathcal{C}}$ learns a share $\tilde{s}_j$ of the N values, and a piece of verification information $\widetilde{\mathsf{aux}}$ is made public.
- $\{F_k(s)\}_{k=1}^N \leftarrow \mathsf{DPFSS.R}(\{\tilde{\mathbf{s}}_j\}_{j=1}^n, \widetilde{\mathsf{aux}})$. The *reconstruction* algorithm takes $\{\tilde{\mathbf{s}}_j\}_{j=1}^n$ and $\widetilde{\mathsf{aux}}$ as input, and outputs the N evaluation results $\{F_k(s)\}_{k=1}^N$.

In our model (Fig. 2), the functions $\{F_k\}_{k=1}^N$ are *public* to both committees and depend on the application scenarios.

Adversary: In our model, an adversary is a PPT algorithm that may corrupt at most $t(< n/3)$ parties in every committee ($\leq t$ parties in $\mathcal{C}$ and $\leq t$ parties in $\tilde{\mathcal{C}}$), in order to learn information about the secret vector $\mathbf{s}$ or the evaluation result $\{F_k(\mathbf{s})\}_{k=1}^N$, or fool the output client to output a wrong evaluation result. The adversary $\mathcal{A}$ may be mobile and active:

- *Mobile.* $\mathcal{A}$ may corrupt some parties and later moves to corrupt the others so that it can eventually learn all information held by the corrupted parties.
- *Active.* $\mathcal{A}$ can modify the internal states as well as control the behaviors of the corrupted parties.

Network: We assume a *synchronous* network. The input client P_{in}, the output client P_{out} and the parties in committees $\mathcal{C}$ and $\tilde{\mathcal{C}}$ all have access to secure point-to-point (P2P) channels such that they know the source of the received message. The adversary cannot learn the content of the message or speak on behalf of an honest party. Furthermore, there is a *broadcast* channel via which any message can be made public to all involved parties in our DPFSS model.

Definition 2 (Integrity and Secrecy): The scheme *DPFSS* should satisfy the properties of integrity and secrecy with respect to the adversary and the network described as above.

- **Integrity.** In the presence of any mobile and active PPT adversary $\mathcal{A}$ that controls $\leq t$ parties from $\mathcal{C}$ and $\leq t$ parties from $\tilde{\mathcal{C}}$, the output client always outputs the correct values of the N evaluation results $\{F_k(\mathbf{s})\}_{k=1}^N$, by faithfully executing $\mathsf{DPFSS.R}$.
- **Secrecy.** Any mobile and active PPT adversary $\mathcal{A}$ that controls $\leq t$ parties from $\mathcal{C}$ and $\leq t$ parties from $\tilde{\mathcal{C}}$ learns no information about the secret vector s or the N evaluation results $\{F_k(\mathbf{s})\}_{k=1}^N$.

Remark 1: Similar to [18], the integrity and secrecy and of our DPFSS are informally defined. Proofs of these properties are quite formal and depend on the design idea that every message in our scheme will be verified and the sender of a maliciously designed message will be removed from the protocol execution immediately.

IV. DPFSS FOR QUADRATIC FUNCTIONS

In this section, we construct a DPFSS scheme that allows the old committee $\mathcal{C}$ to pass the evaluation results $\{F_k(\mathbf{s})\}_{k=1}^N$ of N quadratic functions $F_1, \dots, F_N$ at the secret vector $s = (s_1, \dots, s_m)$ to a new committee $\tilde{\mathcal{C}}$, where the function F_k ($k \in [N]$) has the following form:

$$F_k(\mathbf{s}) = a_0^{(k)} + \sum_{\iota_1 \in [m]} a_{\iota_1}^{(k)} s_{\iota_1} + \sum_{\iota_1, \iota_2 \in [m]} a_{\iota_1, \iota_2}^{(k)} s_{\iota_1} s_{\iota_2}. \quad (4)$$

A. SHARE GENERATION

In the share generation phase (Protocol 1) of our DPFSS scheme, the input client P_{in} first generates a bilinear group context $\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ and uses the cyclic group $\mathbb{G}$ to set up a commitment key $\mathsf{ck} = (\mathbb{G}, g)$ for Feldman's polynomial commitment [20]; then it invokes Shamir's secret sharing to choose a polynomial f_i and use f_i to compute a sharing $[s_i]_t$ for every element s_i of the secret vector $\mathbf{s}$; finally, it commits to f_i with Feldman's polynomial commitment scheme.

Protocol 1. Share Generation $\mathsf{DPFSS.S}$

Input: security parameter κ, threshold t, committee size $n(> 3t)$, secrets $\mathbf{s} = (s_1, \dots, s_m)$.
Output: $\{\mathbf{s}_j\}_{j=1}^n$, $\mathsf{aux} = (\mathsf{bp}, \mathsf{ck}, \{\mathsf{com}_{f_i}\}_{i=1}^m)$.

```
1: Invoke $\mathsf{FC.Setup}(\kappa)$ by running $\mathcal{BG}(\kappa)$ to generate a bilinear group context $\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ and defining a commitment key $\mathsf{ck} = (\mathbb{G}, g)$.
2: For every $i \in [m]$, invoke $\mathsf{SSS.Share}(t, n, s_i)$ to choose a polynomial $f_i(X) = s_i + f_{i,1}X + \cdots + f_{i,t}X^t$ and compute a sharing $[s_i]_t = (s_{i,1}, \dots, s_{i,n}) = (f_i(1), \dots, f_i(n))$ of $s_i$. Set $\mathbf{s}_j = (s_{1,j}, \dots, s_{m,j})$ for every $j \in [n]$.
3: For every $i \in [m]$, invoke $\mathsf{FC.Commit}(\mathsf{ck}, f_i)$ to generate a commitment $\mathsf{com}_{f_i} = (g^{s_i}, g^{f_{i,1}}, \dots, g^{f_{i,t}})$.
```

B. FUNCTION HANDOFF

The function handoff phase (Protocol 2) contains two subphases: $\mathsf{Prepare}$ and $\mathsf{Refresh}$ (Fig. 3). In $\mathsf{Prepare}$, the parties in $\tilde{\mathcal{C}}$ jointly generate N *Random coupled sharings* (RCSs) $([r_k]_t, [\tilde{r}_k]_t)_{k=1}^N$, where each $([r_k]_t, [\tilde{r}_k]_t)$ consists two degree-t sharings of the same random value $r_k(= \tilde{r}_k)$, which are distributed to the old and new committee, respectively.

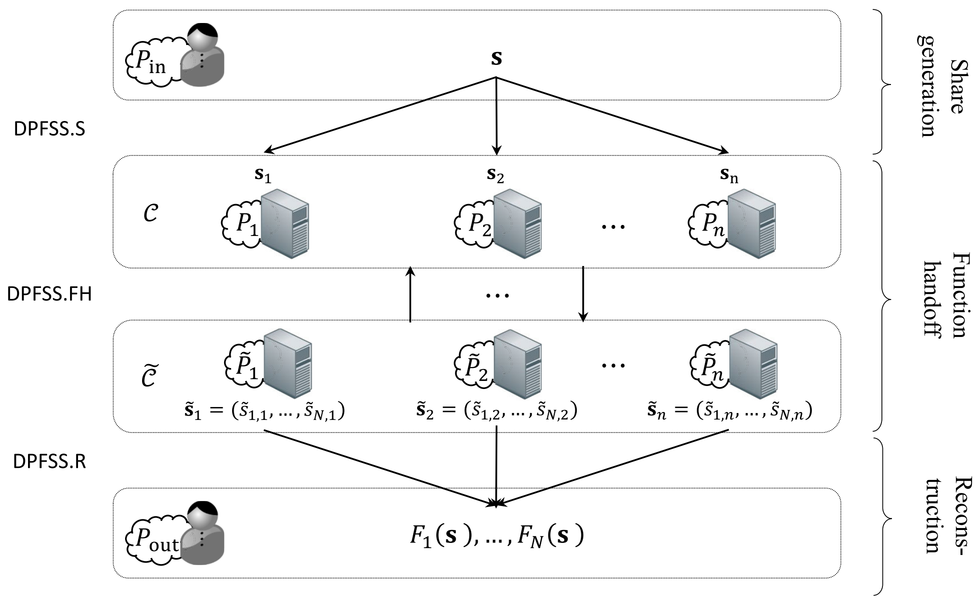

$P_{\rm in}$
$\mathbf{s}$
DPFSS.S
Share generation
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_n$
$\mathcal{C}$
P_1
P_2
P_n
DPFSS.FH
Function handoff
$\tilde{\mathcal{C}}$
$\tilde{P}_1$
$\tilde{P}_2$
$\tilde{P}_n$
$\tilde{\mathbf{s}}_1 = (\tilde{s}_{1,1}, \ldots, \tilde{s}_{N,1})$
$\tilde{\mathbf{s}}_2 = (\tilde{s}_{1,2}, \ldots, \tilde{s}_{N,2})$
$\tilde{\mathbf{s}}_n = (\tilde{s}_{1,n}, \ldots, \tilde{s}_{N,n})$
DPFSS.R
Reconstruction
$P_{\rm out}$
$F_1(\mathbf{s}), \ldots, F_N(\mathbf{s})$

FIGURE 2. Our DPFSS model.

In the refresh phase, each RCS $([r_k]_t, [\tilde{r}_k]_t)$ is used to transfer a sharing of $F_k(\mathbf{s})$ from the old committee $\mathcal{C}$ to a new committee $\tilde{\mathcal{C}}$.

Protocol 2. Function Handoff DPFSS.FH

Participants: old committee $\mathcal{C} = \{P_1, \ldots, P_n\}$, new committee $\tilde{\mathcal{C}} = \{\tilde{P}_1, \ldots, \tilde{P}_n\}$

Input: $\{F_k\}_{k=1}^N$, $\{\mathbf{s}_j\}_{j=1}^n$, aux $=$ (bp, ck, $\{\mathsf{com}_{f_i}\}_{i=1}^m$), where $\{F_k\}_{k=1}^N$ and aux are public inputs known to all participants and s_j is a private input of P_j for all $j \in [n]$.

Output: $\{\tilde{\mathbf{s}}_j\}_{j=1}^n$, $\widetilde{\mathsf{aux}} = (\mathsf{ck}, \{\mathsf{com}_{[\tilde{s}_k]_t}\}_{k=1}^N)$.

The prepare phase:

1: Every party in $\mathcal{C} \cup \tilde{\mathcal{C}}$ initializes $B = \emptyset$ as the set of corrupted parties.
2: Every party $\tilde{P}_i$ randomly chooses two polynomials $h_i(X) = u_i + \sum_{z=1}^t h_{i,z}X^z$ and $\tilde{h}_i(X) = \tilde{u}_i + \sum_{z=1}^t \tilde{h}_{i,z}X^z$, where $u_i = \tilde{u}_i$. and prepares a RCS $([u_i]_t, [\tilde{u}_i]_t)$ such that $[u_i]_t = (u_{i,1}, \ldots, u_{i,n})$, $[\tilde{u}_i]_t = (\tilde{u}_{i,1}, \ldots, \tilde{u}_{i,n})$. It sends $u_{i,j}$ and $\tilde{u}_{i,j}$ to P_j and $\tilde{P}_j$ respectively for every $j \in [n]$, and then broadcasts $\mathsf{com}_{h_i} = \mathsf{FC.Commit}(\mathsf{ck}, h_i)$ and $\mathsf{com}_{\tilde{h}_i} = \mathsf{FC.Commit}(\mathsf{ck}, \tilde{h}_i)$.
3: Every party P_j computes $b = \mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{h_i}, j, u_{i,j})$ to check the correctness of $u_{i,j}$. If $b = 0$, $\tilde{P}_i$ is required to publish a share $u'_{i,j}$. If $\mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{h_i}, j, u'_{i,j}) = 0$, $\tilde{P}_i$ is found corrupted and all parties in $\mathcal{C} \cup \tilde{\mathcal{C}}$ set $B \leftarrow B \cup \{\tilde{P}_i\}$; otherwise, all parties set $u_{i,j} = u'_{i,j}$.
4: Every party $\tilde{P}_j$ computes $b = \mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{\tilde{h}_i}, j, \tilde{u}_{i,j})$ to check the correctness of $\tilde{u}_{i,j}$. If $b = 0$, $\tilde{P}_i$ is required to publish a share $\tilde{u}'_{i,j}$. If $\mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{\tilde{h}_i}, j, \tilde{u}'_{i,j}) = 0$, $\tilde{P}_i$ is found corrupted and all parties in $\mathcal{C} \cup \tilde{\mathcal{C}}$ set $B \leftarrow B \cup \{\tilde{P}_i\}$; otherwise, all parties set $\tilde{u}_{i,j} = \tilde{u}'_{i,j}$.
5: For every $i \in [n]$, all parties in $\mathcal{C} \cup \tilde{\mathcal{C}}$ check if $\mathsf{com}_{h_i}[1] = \mathsf{com}_{\tilde{h}_i}[1]$. If not, they set $B \leftarrow B \cup \{\tilde{P}_i\}$.
6: For every $i \in B$, set $[u_i]_t = [\tilde{u}_i]_t = 0_n$, and $\mathsf{com}_{h_i} = \mathsf{com}_{\tilde{h}_i} = 1_{t+1}$.
7: Let $\mathbf{M} = (M_{k,z}) = (k^{z-1}) \in \mathbb{F}_p^{N \times n}$ be a public Vandermonde matrix. For every $k \in [N]$, P_j and $\tilde{P}_j$ compute $r_{k,j}$ and $\tilde{r}_{k,j}$, respectively, where

$$r_{k,j} = \mathbf{M}_{k,1}u_{1,j} + \cdots + \mathbf{M}_{k,n}u_{n,j},$$
$$\tilde{r}_{k,j} = \mathbf{M}_{k,1}\tilde{u}_{1,j} + \cdots + \mathbf{M}_{k,n}\tilde{u}_{n,j}. \tag{5}$$

The commitments of $[r_k]_t = (r_{k,j})_{j=1}^n$ and $[\tilde{r}_k]_t = (\tilde{r}_{k,j})_{j=1}^n$ are computed as

$$\mathsf{com}_{[r_k]_t} = (\mathsf{com}_{h_1})^{\mathbf{M}_{k,1}} \circ \cdots \circ (\mathsf{com}_{h_n})^{\mathbf{M}_{k,n}},$$
$$\mathsf{com}_{[\tilde{r}_k]_t} = (\mathsf{com}_{\tilde{h}_1})^{\mathbf{M}_{k,1}} \circ \cdots \circ (\mathsf{com}_{\tilde{h}_n})^{\mathbf{M}_{k,n}}. \tag{6}$$

The refresh phase:

1: For all $k \in [N]$, every party P_j computes $v_{k,j} = r_{k,j} + F_k(\mathbf{s}_j) = h_k(j) + F_k(f_1(j), \ldots, f_m(j))$. It then publishes a vector $\mathbf{v}_j = (v_{1,j}, \ldots, v_{N,j})$.
2: For all $(k, j) \in [N] \times [n]$, every party in $\tilde{\mathcal{C}}$ checks the correctness of $v_{k,j}$ with

$$e(g,g)^{v_{k,j}} = e(g^{r_{k,j}}, g) \cdot e(g,g)^{a_0^{(k)}} \cdot \prod_{\iota_1 \in [m]} e(g^{s_{\iota_1,j}}, g)^{a_{\iota_1}^{(k)}} \cdot \prod_{\iota_1,\iota_2 \in [m]} e(g^{s_{\iota_1,j}}, g^{s_{\iota_2,j}})^{a_{\iota_1\iota_2}^{(k)}}, \tag{7}$$

where the witnesses of $r_{k,j}$ and every $\{s_{\iota,j}\}_{\iota \in [m]}$ are computed from the commitments $\mathsf{com}_{[r_k]_t}$, com_{f_ι} as

$$g^{r_{k,j}} = \prod_{z=0}^{t} \left(\mathsf{com}_{[r_k]_t}[z+1]\right)^{j^z},$$
$$g^{s_{\iota,j}} = \prod_{z=0}^{t} \left(\mathsf{com}_{f_\iota}[z+1]\right)^{j^z}.$$

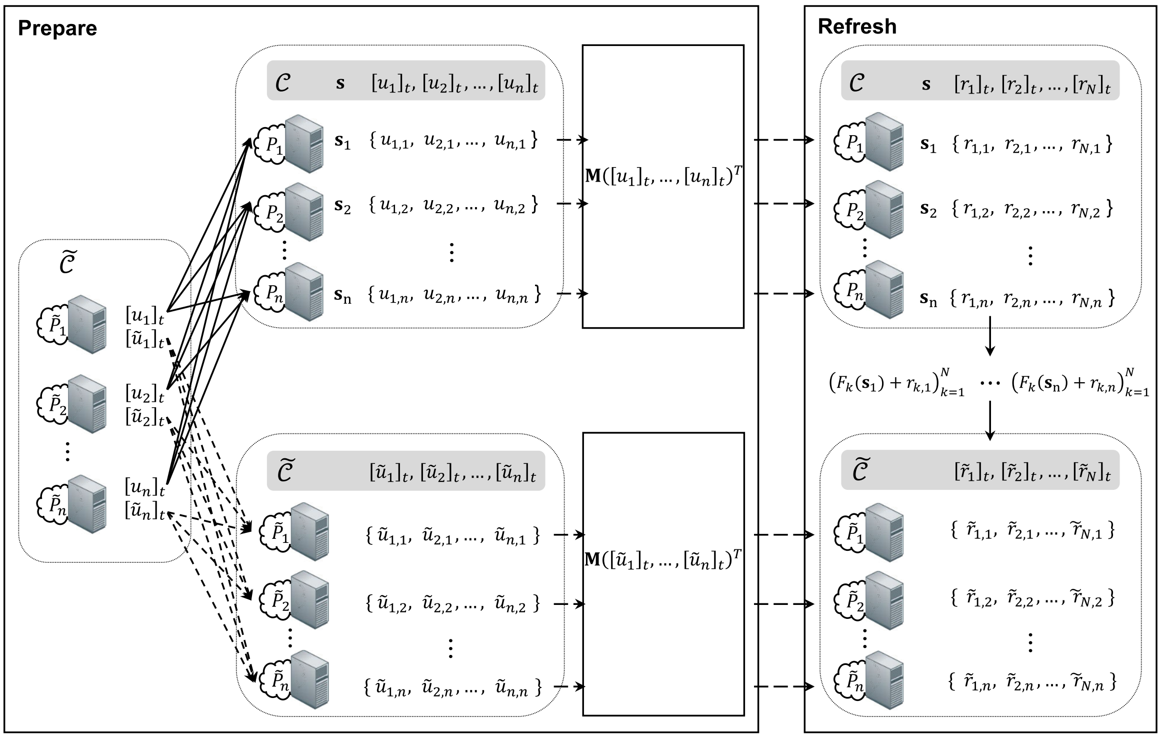

Prepare
Refresh
$\mathcal{C}$
$\mathbf{s}$ $[u_1]_t, [u_2]_t, \dots, [u_n]_t$
$\mathbf{s}_1$ $\{u_{1,1}, u_{2,1}, \dots, u_{n,1}\}$
$\mathbf{s}_2$ $\{u_{1,2}, u_{2,2}, \dots, u_{n,2}\}$
$\mathbf{s}_n$ $\{u_{1,n}, u_{2,n}, \dots, u_{n,n}\}$
$\mathbf{M}([u_1]_t, \dots, [u_n]_t)^T$
$\tilde{\mathcal{C}}$
$[u_1]_t$ $[\tilde{u}_1]_t$
$[u_2]_t$ $[\tilde{u}_2]_t$
$[u_n]_t$ $[\tilde{u}_n]_t$
$[\tilde{u}_1]_t, [\tilde{u}_2]_t, \dots, [\tilde{u}_n]_t$
$\{\tilde{u}_{1,1}, \tilde{u}_{2,1}, \dots, \tilde{u}_{n,1}\}$
$\{\tilde{u}_{1,2}, \tilde{u}_{2,2}, \dots, \tilde{u}_{n,2}\}$
$\{\tilde{u}_{1,n}, \tilde{u}_{2,n}, \dots, \tilde{u}_{n,n}\}$
$\mathbf{M}([\tilde{u}_1]_t, \dots, [\tilde{u}_n]_t)^T$
$\mathbf{s}$ $[r_1]_t, [r_2]_t, \dots, [r_N]_t$
$\mathbf{s}_1$ $\{r_{1,1}, r_{2,1}, \dots, r_{N,1}\}$
$\mathbf{s}_2$ $\{r_{1,2}, r_{2,2}, \dots, r_{N,2}\}$
$\mathbf{s}_n$ $\{r_{1,n}, r_{2,n}, \dots, r_{N,n}\}$
$(F_k(\mathbf{s}_1) + r_{k,1})_{k=1}^N \cdots (F_k(\mathbf{s}_n) + r_{k,n})_{k=1}^N$
$[\tilde{r}_1]_t, [\tilde{r}_2]_t, \dots, [\tilde{r}_N]_t$
$\{\tilde{r}_{1,1}, \tilde{r}_{2,1}, \dots, \tilde{r}_{N,1}\}$
$\{\tilde{r}_{1,2}, \tilde{r}_{2,2}, \dots, \tilde{r}_{N,2}\}$
$\{\tilde{r}_{1,n}, \tilde{r}_{2,n}, \dots, \tilde{r}_{N,n}\}$
P_1 P_2 P_n $\tilde{P}_1$ $\tilde{P}_2$ $\tilde{P}_n$

FIGURE 3. Share flow of DPFSS.FH. The solid lines represent the flow between different committees and the dotted lines represent the flow in the same committee.

We say that a vector v_j is *valid* if all of its elements pass the above verification. Let $J = \{j \in [n] : v_j \text{ is valid}\}$. For every $k \in [N]$, every party in $\tilde{\mathcal{C}}$ interpolates a degree-$2t$ polynomial $\phi_k(X)$ such that $\phi_k(j) = v_{k,j}$ for all $j \in J$, and computes $v_k = \phi_k(0)$ and $\mathsf{com}_{v_k} = (g^{v_k}, \mathbf{1}_t)$, where v_k is considered as a constant polynomial.

3: For all $k \in [N]$, every party $\tilde{P}_j$ computes $\tilde{s}_{k,j} = v_k - \tilde{r}_{k,j}$. Finally, $\tilde{P}_j$ learns $\tilde{\mathbf{s}}_j = (\tilde{s}_{1,j}, \dots, \tilde{s}_{N,j})$.

4: For every $k \in [N]$, every party in $\tilde{\mathcal{C}}$ publishes the commitment $\mathsf{com}_{[\tilde{s}_k]_t} = \mathsf{com}_{v_k} \oslash \mathsf{com}_{[\tilde{r}_k]_t}$. The parties determine $\mathsf{com}_{[\tilde{s}_k]_t}$ using a majority rule.

C. RECONSTRUCTION

In this phase (Protocol 3), an output client P_{out} reconstructs the functions with the shares provided by the new committee.

Protocol 3. Reconstruction DPFSS.R

Input: $\{\tilde{\mathbf{s}}_j\}_{j=1}^n$, $\widetilde{\mathsf{aux}} = (\mathsf{ck}, \{\mathsf{com}_{[\tilde{s}_k]_t}\}_{k=1}^N)$.
Output: $\tilde{\mathbf{s}} = (\tilde{s}_1, \dots, \tilde{s}_N)$

```
1: For all $k \in [N]$, $P_{out}$ computes $b_j = \mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{[\tilde{s}_k]_t}, j, \tilde{s}_{k,j})$ to check the correctness of each $\tilde{s}_{k,j}$ in $\tilde{\mathbf{s}}_j$. Let $J = \{j \in [n] : b_j = 1\}$. $P_{out}$ applies $\mathsf{SSS.Recon}$ to the shares $\{\tilde{s}_{k,j}\}_{j \in J}$ to reconstruct a polynomial $\psi_k(X)$ of degree $\leq t$ and sets $y_k = \psi_k(0)$.
2: $P_{out}$ eventually outputs a vector $\tilde{\mathbf{s}} = (\tilde{s}_1, \dots, \tilde{s}_N)$.
```

D. SECURITY ANALYSIS

Integrity of Our DPFSS Scheme: As per Definition 2, the integrity of our DPFSS scheme requires that: *In the presence of any mobile and active PPT adversary $\mathcal{A}$ that controls $\leq t$ parties from $\mathcal{C}$ and $\leq t$ parties from $\tilde{\mathcal{C}}$, the output client always outputs the correct values of the N evaluation results $\{F_k(\mathbf{s})\}_{k=1}^N$, by faithfully executing* DPFSS.R.

Without loss of generality, we suppose that $\mathcal{A}$ is a PPT adversary that controls exactly t parties from $\mathcal{C}$ (say $P_1, \dots, P_t$) and t parties from $\tilde{\mathcal{C}}$ (say $\tilde{P}_1, \dots, \tilde{P}_t$) throughout the entire execution of DPFSS = (DPFSS.S, DPFSS.FH, DPFSS.R). Note that $\mathcal{A}$ sends no messages in the share generation phase DPFSS.S. It suffices to show that $\mathcal{A}$ cannot fool the output client into outputting a vector $\tilde{\mathbf{s}} \neq \{F_k(\mathbf{s})\}_{k=1}^N$ by participating DPFSS.FH and DPFSS.R. Our proof of integrity is divided into three parts, which are regarding the prepare phase of DPFSS.FH, the refresh phase of DPFSS.FH, and the reconstruction protocol DPFSS.R, respectively.

Lemma 1: Let $\mathcal{A}$ be a PPT adversary that controls the parties $P_1, \dots, P_t \in \mathcal{C}$ and $\tilde{P}_1, \dots, \tilde{P}_t \in \tilde{\mathcal{C}}$. At the end of the prepare phase of DPFSS.FH, there exist $2N$ polynomials $\{\sigma_k(X)\}_{k=1}^N$ and $\{\tilde{\sigma}_k(X)\}_{k=1}^N$ of degree $\leq t$ such that for all $k \in [N]$,

- $\sigma_k(0) = \tilde{\sigma}_k(0)$;
- $\sigma_k(j) = r_{k,j}, \forall j \in \{t+1, \dots, n\}$;
- $\tilde{\sigma}_k(j) = \tilde{r}_{k,j}, \forall j \in \{t+1, \dots, n\}$.

Proof: For every $i \in [n]$, let $[u_i]_t = (u_{i,1}, \dots, u_{i,n})$ and $[\tilde{u}_i]_t = (\tilde{u}_{i,1}, \dots, \tilde{u}_{i,n})$ be the RCS after the steps 3, 4, 5, and 6 of the prepare phase of DPFSS.FH have been executed. Note that this RCS $([u_i]_t, [\tilde{u}_i]_t)$ may be different from the RCS at step 2 of the prepare phase, which may undergo changes at steps 3, 4, 5, and 6. We start the proof by showing that there exist $2n$ polynomials $\{\tau_i(X)\}_{i=1}^n$ and $\{\tilde{\tau}_i(X)\}_{i=1}^n$ such that for every $i \in [n]$:

- $\tau_i(0) = \tilde{\tau}_i(0)$;
- $\tau_i(j) = u_{i,j}, \forall j \in \{t+1, \dots, n\}$;
- $\tilde{\tau}_i(j) = \tilde{u}_{i,j}, \forall j \in \{t+1, \dots, n\}$.

For every $i \in B$, the party $\tilde{P}_i$ is found corrupted. Both $[u_i]_t$ and $[\tilde{u}_i]_t$ have been set to 0_n at step 6. Thus, we can take $\tau_i(X) = \tilde{\tau}_i(X) = 0$. For every $i \in [n] \setminus B$, the specifications of steps 3, 4, 5 and 6 show that we must have $\mathsf{com}_{h_i}[1] = \mathsf{com}_{\tilde{h}_i}[1]$ and $\mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{h_i}, j, u_{i,j}) = \mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{\tilde{h}_i}, j, \tilde{u}_{i,j}) = 1$. It suffices to set $\tau_i(X) = h_i(X)$ and $\tilde{\tau}_i(X) = \tilde{h}_i(X)$. For all $k \in [N]$, let

$$\sigma_k(X) = \sum_{i=1}^{n} M_{k,i} \cdot \tau_i(X), \quad \tilde{\sigma}_k(X) = \sum_{i=1}^{n} M_{k,i} \cdot \tilde{\tau}_i(X). \qquad (8)$$

Then the specifications of step 7 show that these polynomials have degree $\leq t$ and satisfy the expected properties.

Lemma 2: Let $\mathcal{A}$ be a PPT adversary that controls the parties $P_1, \dots, P_t \in \mathcal{C}$ and $\tilde{P}_1, \dots, \tilde{P}_t \in \tilde{\mathcal{C}}$. At the end of the refresh phase of $\mathsf{DPFSS.FH}$, there exist N polynomials $\{\psi_k(X)\}_{k=1}^N$ of degree $\leq t$ such that $\psi_k(0) = F_k(\mathbf{s})$ for all $k \in [N]$.

Proof: For every $j \in [n]$, P_j learns a share $\mathbf{s}_j$ of $\mathbf{s}$ from the share generation phase $\mathsf{DPFSS.S}$ and N shares $\{r_{1,j}, \dots, r_{N,j}\}$ of the sharings $\{[r_k]_t\}_{k=1}^N$ from the prepare phase of $\mathsf{DPFSS.FH}$. Let $\mathbf{v}_j = (v_{1,j}, \dots, v_{N,j})$ be the vector published by P_j at step 1 of the refresh phase of $\mathsf{DPFSS.FH}$.

By Lemma 1, for all $k \in [N]$ and $j \in \{t+1, \dots, n\}$, $v_{k,j} = r_{k,j} + F_k(\mathbf{s}_j)$ is published by an honest party P_j and $v_{k,j} = \sigma_k(j) + F_k(f_1(j), \dots, f_m(j))$ is the evaluation of the polynomial

$$\phi_k(X) = \sigma_k(X) + F_k(f_1(X), \dots, f_m(X))$$

at $X = j$, where the polynomial has degree $\leq 2t$ and $f_1, \dots, f_m$ are the m polynomials of degree $\leq t$ from $\mathsf{DPFSS.S}$. In particular, the equation $v_{k,j} = r_{k,j} + F_k(\mathbf{s}_j)$ implies that the Eq. (7) at step 2 of the refresh phase is true. Thus, v_j is valid and $j \in J$.

On the other hand, for $j \in [t]$, P_j is controlled by $\mathcal{A}$ and we may not have $\phi_k(j) = v_{k,j}$, which holds if and only if the Eq. (7) holds. If $\phi_k(j) = v_{k,j}$ is true for all $k \in [N]$, then v_j will be valid and $j \in J$.

The above analysis shows that $|J| \geq n - t$. As stated in $\mathsf{DPFSS.S}$, we have $n > 3t$ and so $|J| > 2t$. For every $k \in [N]$, $\phi_k(X)$ can be interpolated with the $\geq 2t+1$ valid shares $\{v_{k,j}\}_{j \in J}$. In particular, we have

$$v_k = \phi_k(0) = \sigma_k(0) + F_k(\mathbf{s}).$$

For all $k \in [N]$ and $j \in \{t+1, \dots, n\}$, the honest party $\tilde{P}_j$ computes a value $\tilde{s}_{k,j} = v_k - \tilde{r}_{k,j}$, which is equal to $\phi_k(0) - \tilde{\sigma}_k(j) = \sigma_k(0) + F_k(\mathbf{s}) - \tilde{\sigma}_k(j)$, an evaluation of the polynomial

$$\psi_k(X) = \sigma_k(0) + F_k(\mathbf{s}) - \tilde{\sigma}_k(X) \qquad (9)$$

at $X = j$. By Lemma 1, $\psi_k(X)$ has degree $\leq t$ and $\psi_k(0) = F_k(s)$.

For all $k \in [N]$, the honest parties $\tilde{P}_{t+1}, \dots, \tilde{P}_n$ compute the same commitment $\mathsf{com}_{[\tilde{s}_k]_t} = \mathsf{com}_{v_k} \oslash \mathsf{com}_{[\tilde{r}_k]_t}$, because they all have the same v_k and $\mathsf{com}_{[\tilde{r}_k]_t}$. In fact, $\mathsf{com}_{[\tilde{s}_k]_t}$ is a commitment of $\psi_k(X)$. As the majority of the parties in $\tilde{\mathcal{C}}$ are honest, this commitment will be output as an element of $\widetilde{\mathsf{aux}}$.

Lemma 3: Let $\mathcal{A}$ be a PPT adversary that controls the parties $P_1, \dots, P_t \in \mathcal{C}$ and $\tilde{P}_1, \dots, \tilde{P}_t \in \tilde{\mathcal{C}}$. At the end of the reconstruction phase $\mathsf{DPFSS.R}$, the output client P_{out} can reconstruct $F_1(\mathbf{s}), \dots, F_N(\mathbf{s})$.

Proof: By the proof of Lemma 2, for all $k \in [N]$ and $j \in \{t+1, \dots, n\}$, $\tilde{s}_{k,j}$ is the evaluation the polynomial $\psi_k(X)$, whose commitment is $\mathsf{com}_{[\tilde{s}_k]_t}$. Therefore, $\mathsf{FC.Verify}(\mathsf{ck}, \mathsf{com}_{[\tilde{s}_k]_t}, j, \tilde{s}_{k,j}) = 1$ for all $k \in [N]$ and $j \in \{t+1, \dots, n\}$. Thus, $\{t+1, \dots, n\} \subseteq J$ and J contains $\geq t+1$ elements. $\mathsf{DPFSS.R}$ will reconstruct the polynomial $\psi_k(X)$ of Eq. (9) for all $k \in [N]$. Consequently, P_{out} will output $\tilde{\mathbf{s}} = (\tilde{s}_1, \dots, \tilde{s}_N) = (\psi_1(0), \dots, \psi_N(0)) = (F_1(s), \dots, F_N(s))$.

Theorem 1 (Integrity): In the presence of any mobile and active PPT adversary $\mathcal{A}$ that controls $\leq t$ parties from $\mathcal{C}$ and $\leq t$ parties from $\tilde{\mathcal{C}}$, the output client will output the correct N functions $\{F_k(\mathbf{s})\}_{k=1}^N$ by faithfully executing $\mathsf{DPFSS.R}$.

Proof: The secrets $\mathbf{s}$ are firstly shared among the old committee $\mathcal{C}$ by an input client. By Lemma 1 and Lemma 2, N functions $\{F_k(\mathbf{s})\}_{k=1}^N$ are transferred from $\mathcal{C}$ to $\tilde{\mathcal{C}}$ with N RCSs in the function handoff phase. By Lemma 3, these functions stay invariant in the reconstruction phase. Therefore an honest output client will output $\{F_k(\mathbf{s})\}_{k=1}^N$. We conclude that the integrity of our DPFSS scheme holds throughout the execution.

Secrecy of Our DPFSS Scheme: As per Definition 2, the secrecy of our DPFSS scheme requires that: *Any mobile and active PPT adversary $\mathcal{A}$ that controls $\leq t$ parties from $\mathcal{C}$ and $\leq t$ parties from $\tilde{\mathcal{C}}$ learns no information about the secret vector s or the N evaluation results $\{F_k(\mathbf{s})\}_{k=1}^N$.*

Without loss of generality, we suppose that $\mathcal{A}$ is a PPT adversary that controls exactly t parties from $\mathcal{C}$ (say $P_1, \dots, P_t$) and t parties from $\tilde{\mathcal{C}}$ (say $\tilde{P}_1, \dots, \tilde{P}_t$) throughout the entire execution of DPFSS. Recall that our DPFSS scheme consists of three phases: share generation $\mathsf{DPFSS.S}$, function handoff $\mathsf{DPFSS.FH}$ and reconstruction $\mathsf{DPFSS.R}$. As the parties controlled by $\mathcal{A}$ receive no messages in $\mathsf{DPFSS.R}$, it suffices to show that $\mathcal{A}$ learns no information about s and $\{F_k(\mathbf{s})\}_{k=1}^N$ in $\mathsf{DPFSS.S}$ and $\mathsf{DPFSS.FH}$. Other than the public inputs, the information obtained by $\mathcal{A}$ in the two phases can be described as follows:

- $\mathsf{DPFSS.S}$: t shares $\{\mathbf{s}_i\}_{i=1}^t$ of the secret vector $\mathbf{s}$ from $\{P_i\}_{i=1}^t$.
- $\mathsf{DPFSS.FH}$: tN shares $\{r_{k,1}, \dots, r_{k,t}\}_{k=1}^N$ of $\{[r_k]_t\}_{k=1}^N$ and n public vectors $\{\mathbf{v}_i\}_{i=1}^n$ from $\{P_i\}_{i=1}^t$; tN shares $\{\tilde{r}_{k,1}, \dots, \tilde{r}_{k,t}\}_{k=1}^N$ of $\{[\tilde{r}_k]_t\}_{k=1}^N$ and t intermediate evaluation results $\{\tilde{\mathbf{s}}_i\}_{i=1}^t$ from $\{\tilde{P}_i\}_{i=1}^t$.

Lemma 4: Let $\mathcal{A}$ be a PPT adversary that controls the parties $P_1, \dots, P_t \in \mathcal{C}$ and $\tilde{P}_1, \dots, \tilde{P}_t \in \tilde{\mathcal{C}}$. In the share generation phase, the adversary $\mathcal{A}$ learns no information about $\mathbf{s}$ from the corrupted parties in $\mathcal{C}$.

Proof: In $\mathsf{DPFSS.S}$, $\mathcal{A}$ obtains t shares $\{\mathbf{s}_i\}_{i=1}^t$ of $\mathbf{s}$ from the corrupted parties $\{P_i\}_{i=1}^t$ and m commitments $\{\mathsf{com}_{f_z}\}_{z=1}^m$.

For every $s_z \in s$ ($z \in [m]$), $\{\mathbf{s}_i\}_{i=1}^t$ contain t shares $s_{z,1}, \ldots, s_{z,t}$ in $[s_z]_t$, which is a Shamir sharing of s_z generated with f_z, a polynomial of degree $\leq t$. The security of Shamir SSS shows that $\mathcal{A}$ learns no information about s_z. Hence, $\mathcal{A}$ learns no information about s from $\{\mathbf{s}_i\}_{i=1}^t$.

On the other hand, for every $z \in [m]$, $f_z(0) = s_z$ is an element of s and com_{f_z} is a commitment of f_z. Due to the *computationally hiding property* of Feldman's polynomial commitment, the PPT adversary $\mathcal{A}$ cannot learn s_z from com_{f_z}.

Lemma 5: Let $\mathcal{A}$ be a PPT adversary that controls the parties $P_1, \ldots, P_t \in \mathcal{C}$ and $\tilde{P}_1, \ldots, \tilde{P}_t \in \tilde{\mathcal{C}}$. In the *function handoff*, the adversary $\mathcal{A}$ learns no information about $\{F_k(\mathbf{s})\}_{k=1}^N$ from the corrupted parties in both committees.

Proof: In the *prepare phase* of the function handoff phase $\mathsf{DPFSS.FH}$ $\mathcal{A}$ obtains tN shares $\{r_{k,1}, \ldots, r_{k,t}\}_{k=1}^N$ of $\{[r_k]_t\}_{k=1}^N$ from $\{P_i\}_{i=1}^t$, tN shares $\{\tilde{r}_{k,1}, \ldots, \tilde{r}_{k,t}\}_{k=1}^N$ of $\{[\tilde{r}_k]_t\}_{k=1}^N$ from $\{\tilde{P}_i\}_{i=1}^t$, and $2N$ commitments $\{\mathsf{com}_{[r_k]_t}, \mathsf{com}_{[\tilde{r}_k]_t}\}_{k=1}^N$. The $([r_k]_t, [\tilde{r}_k]_t)_{k=1}^N$ are computed in the following way

$$([r_1]_t, \cdots, [r_N]_t)^T = \mathbf{M}([u_1]_t, \cdots, [u_n]_t)^T,$$
$$([\tilde{r}_1]_t, \cdots, [\tilde{r}_N]_t)^T = \mathbf{M}([\tilde{u}_1]_t, \cdots, [\tilde{u}_n]_t)^T,$$

where $([u_i]_t, [\tilde{u}_i]_t)_{i=1}^n$ are generated by the parties in $\tilde{\mathcal{C}}$.

Noted that $N \leq n - t$ (See Section III) and $\mathbf{M} \in \mathbb{F}_p^{N\times n}$ is a Vandermonde matrix such that any $N \times N$ submatrix is invertible. As $\tilde{P}_{t+1}, \ldots, \tilde{P}_n$ are $n - t \geq N$ honest parties in $\tilde{\mathcal{C}}$, there is a one-to-one map from any N of the $n - t$ honestly generated RCSs $([u_i]_t, [\tilde{u}_i]_t)_{i=t+1}^n$, which are uniformly distributed, to $([r_k]_t, [\tilde{r}_k]_t)_{k=1}^N$. Therefore, the output couples $([r_k]_t, [\tilde{r}_k]_t)_{k=1}^N$ are uniformly distributed as well.

For every $k \in [N]$, the polynomial $\sigma_k(X)$ (resp. $\tilde{\sigma}_k(X)$) from Lemma 1 satisfies $\sigma_k(j) = r_{k,j}$ (resp. $\tilde{\sigma}_k(j) = \tilde{r}_{k,j}$) for $j = t + 1, \ldots, n$, where $(r_{k,j})_{j=t+1}^n$ (resp. $(\tilde{r}_{k,j})_{j=t+1}^n$) are uniformly distributed. Thus, $\sigma_k(X)$ and $\tilde{\sigma}_k(X)$ are random polynomials of degree $\leq t$ such that $\sigma_k(0) = \tilde{\sigma}_k(0)$. Therefore, the adversary $\mathcal{A}$ learns no information about $\sigma_k(0)$ from the shares $\{r_{k,1}, \ldots, r_{k,t}\}$ and $\{\tilde{r}_{k,1}, \ldots, \tilde{r}_{k,t}\}$.

On the other hand, we have that $\mathsf{com}_{[r_k]_t}[1] = g^{\sigma_k(0)}$ and $\mathsf{com}_{[\tilde{r}_k]_t}[1] = g^{\tilde{\sigma}_k(0)}$. By the computationally hiding property of Feldman's polynomial commitment prevents, the PPT adversary $\mathcal{A}$ cannot learn $\sigma_k(0)$ from $\mathsf{com}_{[r_k]_t}$ and $\mathsf{com}_{[\tilde{r}_k]_t}$.

In the *refresh phase* of function handoff, for each $k \in [N]$, the adversary $\mathcal{A}$ can reconstruct $v_k = \sigma_k(0) + F_k(\mathbf{s})$ from $\{v_{k,i}\}_{i=1}^n$, which are n elements from the vectors $\{v_i\}_{i=1}^n$ published by the parties in $\mathcal{C}$ (by Lemma 2). Since $\mathcal{A}$ cannot recover $\sigma_k(0)$, it cannot calculate $F_k(\mathbf{s})$ from v_k.

Besides $\{v_k\}_{k=1}^N$, $\mathcal{A}$ also obtains $\{\tilde{\mathbf{s}}_i = (\tilde{s}_{1,i}, \ldots, \tilde{s}_{N,i})\}_{i=1}^t$ from $\{\tilde{P}_i\}_{i=1}^t$ in the refresh phase, where $\tilde{s}_{k,i} = v_k - \tilde{r}_{k,i}$ for all $k \in [N]$ and $i \in [t]$. Therefore, for every $k \in [N]$, $\mathcal{A}$ has t shares $\{\tilde{s}_{k,1}, \ldots, \tilde{s}_{k,t}\}$ in $[\tilde{s}_k]_t$, whose sharing polynomial is the $\psi_k(X)$ in Eq. (9). Since $\psi_k(X)$ has degree $\leq t$, the t shares do not suffice to interpolate $\psi_k(X)$ and thus give no information about $\psi_k(0) = F_k(\mathbf{s})$.

TABLE 1. Communication complexity of each phase.

Phase		Communication complexity
DPFSS.S		$\mathcal{O}((n+t)m)$
DPFSS.FH	prepare	$\mathcal{O}(n^2 + nt)$
	refresh	$\mathcal{O}((n+t)N)$
DPFSS.R		$\mathcal{O}(Nn)$

Theorem 2 (Secrecy): The mobile and active PPT adversary $\mathcal{A}$, which corrupts $\leq t$ parties in $\mathcal{C}$ and $\leq t$ parties in $\tilde{\mathcal{C}}$, learns no information about $\mathbf{s}$ or $\{F_k(\mathbf{s})\}_{k=1}^N$ during the execution of DPFSS scheme.

Proof: By Lemma 4 and Lemma 5, the adversary $\mathcal{A}$ learns no information of the secrets $\mathbf{s}$ or $\{F_k(\mathbf{s})\}_{k=1}^N$ during the share generation phase and the function handoff phase. And in the reconstruction phase, it is an honest output client who reconstructs $\{F_k(\mathbf{s})\}_{k=1}^N$. Therefore, the secrecy of our DPFSS scheme holds throughout the execution.

V. PERFORMANCE EVALUATION

Like [17] and [18], the efficiency of our DPFSS scheme DPFSS can be measured with its communication complexity and time complexity. In this section, we show that the proposed scheme achieves a communication complexity of $\mathcal{O}(n^2+mn)$ and a time complexity of $O(n)$, which are comparable with the most efficient DPSS scheme from [17]. We also implement DPFSS and evaluate its performance concretely.

A. EFFICIENCY ANALYSIS

Communication Complexity: In every phase of our DPFSS scheme, the communication cost is incurred by the following kinds of operations, whose complexity can be described as follows:

- *Share a secret:* Share a secret s among a committee of n parties requires one to send each party a share in $[s]_t = (s_1, \ldots, s_n)$, which incurs a communication cost of $\mathcal{O}(n)$ field elements.
- *Publish a commitment:* Each commitment consists of $t + 1$ group elements, the communication cost incurred by publishing a commitment is $\mathcal{O}(t)$ group elements.
- *Reconstruct a secret:* Reconstructing a secret s from its sharing $[s]_t$ requires one to obtain n shares from the parties and thus incurs a communication cost of $\mathcal{O}(n)$ field elements.

Based on the above analysis, the communication complexity (see Table 1) of the three phases $\mathsf{DPFSS.S}$, $\mathsf{DPFSS.FH}$ and $\mathsf{DPFSS.R}$ of our scheme can be analyzed as follows:

- $\mathsf{DPFSS.S}$: In this phase, a constant number m of secrets $\mathbf{s}$ are shared by the input client P_{in} and the commitments of the sharings are published. The communication complexity of this phase is $\mathcal{O}(mn)$ field elements plus $\mathcal{O}(mt)$ group elements.

TABLE 2. Time complexity of each phase.

Phase		Time complexity
DPFSS.S		$\mathcal{O}(mnt)$ and $\mathcal{O}(mt)$-exp
DPFSS.FH	prepare	$\mathcal{O}((N+t)n)$ and $\mathcal{O}(nt^2+Nnt)$-exp
	refresh	$\mathcal{O}((m^2+t^2)N)$ and $\mathcal{O}(Nnt^2)$-exp and $\mathcal{O}(Nn)$-bp
DPFSS.R		$\mathcal{O}(Nt^2)$

- DPFSS.FH: In the *prepare* phase, parties in $\tilde{\mathcal{C}}$ generate n RCS $\{([u_i]_t, [\tilde{u}_i]_t)\}_{i=1}^n$, which contain $2n$ sharings, and publish their commitments. The communication cost is $\mathcal{O}(n^2)$ field elements plus $\mathcal{O}(nt)$ group elements. In the *refresh* phase, every $P_j \in \mathcal{C}$ publishes a vector $\mathbf{v}_j$ of N elements. For $k \in [N]$, a sharing $[\tilde{s}_k]_t$ is computed among $\tilde{\mathcal{C}}$ and every $\tilde{P}_j \in \tilde{\mathcal{C}}$ publishes a commitment $\mathsf{com}_{[\tilde{s}_k]_t}$ at the end of this phase. The communication cost is $\mathcal{O}(Nn)$ field elements plus $\mathcal{O}(Nt)$ group elements. Hence, the total communication complexity of DPFSS.FH is $\mathcal{O}(n^2+Nn)$ field elements plus $\mathcal{O}((n+N)t)$ group elements.
- DPFSS.R: Reconstructing the evaluation results requires the parties in the new committee $\tilde{\mathcal{C}}$ to send vectors $\{\tilde{\mathbf{s}}_j\}_{j=1}^n$ to the output client, where each $\mathbf{v}_j$ includes N elements. Hence, the communication complexity is $\mathcal{O}(Nn)$ field elements.

The communication complexity of DPFSS is $\mathcal{O}(n^2+(N+m+t)n+(m+N)t)$, As mentioned in Section III, $N < n-t$ and $t < n/3$. Thus, the communication complexity of our scheme is $\mathcal{O}(n^2+mn)$ field/group elements.

Time Complexity: Let a-exp and a-bp be the time cost of computing a exponentiations in $\mathbb{G}$ and computing a pairings, respectively. In every phase of our DPFSS scheme, the time cost is incurred by the following kinds of operations, whose complexity can be described as follows:

- *Generate a secret sharing:* A secret sharing $[s_i]_t = (s_{i,1}, \ldots, s_{i,n}) = (f_i(1), \ldots, f_i(n)) = (s_{i,1}, \ldots, s_{i,n})$ is a vector of n evaluations of a polynomial $f_i(X) = s + \sum_{j=1}^t f_{i,j}X^i$. The cost of generating $[s_i]_t$ is $\mathcal{O}(nt)$ with Horner's method [49].
- *Commit a secret sharing:* A commitment $\mathsf{com}_{f_s} = (g^{s_i}, g^{f_{i,1}}, \ldots, g^{f_{i,t}})$ of $[s_i]_t$ is vector of $t+1$ exponents of a group generator g. The cost of committing a secret sharing is $\mathcal{O}(t)$-exp
- *Verify a share:* Check whether a given value y is equal to the share $s_{i,j}$ in a sharing $[s_i]_t$, it needs to computes a *witness*
$$\eta_j = \prod_{z=0}^{t} \mathsf{com}_f[z+1]^{j^z} (= g^{s_{i,j}}),$$
then compute g^y and check if the equation $g^y = \eta_j$ holds. The time complexity of verifying a share is $\mathcal{O}(t^2)$-exp.
- *Reconstruct a secret:* Given $t+1$ shares in $[s_i]_t$, the secret s_i can be recovered by applying Lagrange interpolation. The cost of reconstructing a secret is $\mathcal{O}(t^2)$.

The time complexity of our DPFSS is analyzed as follows:

- DPFSS.S: In this phase, the input client P_{in} generates sharings for m secrets $\mathbf{s} = (s_1, \ldots, x_m)$ and computes their commitments. The time complexity of this phase is $\mathcal{O}(nmt)$ field operations plus $\mathcal{O}(mt)$-exp.
- DPFSS.FH: The time complexity of the *prepare* phase is $\mathcal{O}((N+t^2)n)$ field operations plus $\mathcal{O}(nt^2+Nnt)$-exp:
  - In *Step 2*, for all $i \in [n]$, the computation of each $([u_i]_t, [\tilde{u}_i]_t)$ along with the commitments $\mathsf{com}_{h_i}, \mathsf{com}_{\tilde{h}_i}$ costs $\mathcal{O}(nt)$ and $\mathcal{O}(t) - \mathsf{exp}$;
  - In *Step 3-4*, for each $j \in [n]$, the verifications of $\{u_{i,j}\}_{i=1}^n$ (executed by $P_j \in \mathcal{C}$) and the verifications of $\{\tilde{u}_{i,j}\}_{i=1}^n$ (executed by $\tilde{P}_j \in \tilde{\mathcal{C}}$) cost $\mathcal{O}(nt^2)$-exp;
  - In *Step 7*, the computations of $\{([r_k]_t, [\tilde{r}_k]_t)\}_{k=1}^N$, which are N linear combinations of $\{([u_i]_t, [\tilde{u}_i]_t)\}_{i=1}^n$ (See Eq. 5), costs $\mathcal{O}(Nn)$. As shown in Eq. (6), the computation of $\mathsf{com}_{[r_k]_t}$ (resp. $\mathsf{com}_{[\tilde{r}_k]_t}$), which involves component-wise multiplication of $\mathsf{com}_{h_1}, \ldots, \mathsf{com}_{h_n}$ (resp. $\mathsf{com}_{\tilde{h}_1}, \ldots, \mathsf{com}_{\tilde{h}_n}$), costs $\mathcal{O}(Nnt)$-exp.

  The time complexity of the *refresh* phase is $\mathcal{O}((m^2+t^2)N)$ field operations plus $\mathcal{O}(Nnt^2)$-exp plus $\mathcal{O}(Nn)$-bp:
  - In *Step 1*, for each $j \in [n]$ and $k \in [N]$, the evaluation $v_{k,j} = r_{k,j} + F_k(\mathbf{s}_j)$, where $|\mathbf{s}_j| = m$ and F_k is a degree-2 polynomial, costs $\mathcal{O}(Nm^2)$;
  - In *Step 2*, for all $j \in [n]$, the computations of the involved shares' witnesses in each $\mathbf{v}_j = (v_{1,j}, \ldots, v_{N,j})$ have a workload of $\mathcal{O}(Nnt^2)$-exp. For every $k \in [N]$, the verification of the equation $v'_{k,j} = r_{k,j} + F_k(\mathbf{s}_j)$ is done by confirming whether Eq. (7) is true, which costs $\mathcal{O}(Nn)$-bp. The reconstructions of $v_1, \ldots, v_N$ from $2t+1$ valid vectors out of $\{\mathbf{v}_j\}_{j=1}^n$ cost $\mathcal{O}(Nt^2)$.
  - In *Step 3*, for each $j \in [n]$, the computations of $\{\tilde{s}_{k,j} = v_k - \tilde{r}_{k,j}\}_{k=1}^N$ cost $\mathcal{O}(N)$.
  - In *Step 4*, for all $k \in [N]$, the computation of $\mathsf{com}_{[\tilde{s}_k]}$ contains one component-wise division, where every commitment vector has $t+1$ group elements. The cost is $\mathcal{O}(Nt)$-exp;

  Hence, the total time complexity of the function handoff phase is $\mathcal{O}((N+t)n+m^2N+t^2N)$ field operations plus $\mathcal{O}(Nnt^2)$-exp plus $\mathcal{O}(Nn)$-bp.
- DPFSS.R: The output client applies Shamir's SSS to reconstruct every evaluation result, where the Lagrange interpolation is applied. Therefore, the time complexity is $\mathcal{O}(Nt^2)$ field operations.

In conclusion, the time complexity of our DPFSS construction is $\mathcal{O}((N+mt)n+m^2N+t^2N)$ field operations plus $\mathcal{O}(Nnt^2+mt)$-exp plus $\mathcal{O}(Nn)$-bp, which is linear n.

B. EXPERIMENTAL/RESULTS

We implement DPFSS in C++ and use the C library FLINT 2.8.0 [50] to realize large number operations and use the C library RELIC 0.5.0 [51] to realize bilinear

TABLE 3. Time for DPFSS setting with committee sizes 20-100.

Time (sec) \ Phase / Committee size	S	FH1	FH2	R
20	0.001	**0.014**	**2.151**	0.002
40	0.001	**0.050**	**4.304**	0.004
60	0.001	**0.109**	**6.454**	0.007
80	0.002	**0.188**	**8.355**	0.009
100	0.002	**0.294**	**10.413**	0.011

S represents the share generation phase
FH1 represents the prepare phase and FH2 represents the refresh phase
R represents the reconstruction phase

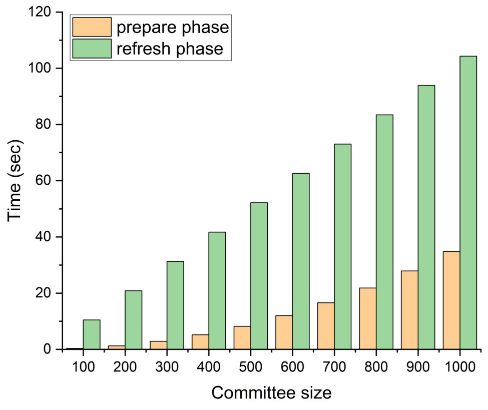


FIGURE 4. Time for DPFSS.FH setting with committee sizes 100-1000.

group operations. Our codes are run in single-thread on an Intel(R) Xeon(R) E-2286G CPU @ 4.0 GHz with 6 cores and 64 GB memory. We run each experiment 100 times and report their average. Our codes are available at: https://github.com/dpfsscode/DPFSS.

Benchmarks: We first choose the parameters for the bilinear group generator $\mathcal{BG}$, Feldman's polynomial commitment scheme FC and our construction DPFSS. Then we quantify the cost of each algorithm or protocol involved in DPFSS.

1) *Parameter selection:* We choose the pairing-friendly elliptic curve BLS12-381 over $\mathbb{F}_q$ for a 128-bit prime q, the order of $\mathbb{G}$, $\mathbb{G}_T$ in the elliptic curve is q. We use the group $\mathbb{G}$ to set up the commitment key of Feldman's scheme FC such that the order of the group is q as well. For our DPFSS, the polynomial arithmetic is done over the polynomial ring $\mathbb{F}_p[X]$ for a 256-bit prime p.
2) *Experimental time complexity:* We implement $\mathcal{BG}$, FC and DPFSS with the parameters stated above. In the optimistic case, where all parties behave honestly, we measure the costs of our DPFSS (DPFSS.S, DPFSS.FH and DPFSS.R) for increasing numbers of nodes in the committee. The results are presented in Table. 3. Compared with the running time of DPFSS.FH, the running time of the other phases is rather tiny. Therefore, we measure the costs of DPFSS.FH for committees with larger sizes. The results are presented in Fig. 4 and it shows that the cost of time matches our expectation of *linear* asymptotic growth in n and suggests a low constant, e.g., for a committee of size $n = 1000$, the total execution time is only about 2 minutes.
3) *Experimental communication complexity:* The data that transmitted throughout our DPFSS includes: mn elements of $\mathbb{F}_p$ (m-share vectors $\{\mathbf{s}_j\}_{j=1}^n$) and m commitments ($\{\mathsf{com}_{f_i}\}_{i=1}^m$), which includes $t+1$ elements of $\mathbb{F}_q$ in the DPFSS.S; n^2+Nn elements of $\mathbb{F}_p$ (shares in $\{([u_i]_t, [\tilde{u}_i]_t)\}_{i=1}^n$, N-share vectors $\{\mathbf{v}_j\}_{j=1}^n$) and $(2+N)n$ commitments ($\{\mathsf{com}_{h_i}, \mathsf{com}_{\tilde{h}_i}\}_{i=1}^n$, $\{\mathsf{com}_{[\tilde{s}_k]_t}\}_{k=1}^N$ from each $\tilde{P}_i \in \tilde{\mathcal{C}}$) in the DPFSS.FH; Nn elements of $\mathbb{F}_p$ (N-share vectors $\{\tilde{\mathbf{s}}_j\}_{j=1}^n$) in the DPFSS.R. In aforementioned parameter selection, prime p is 256-bit and prime q is 254-bit. For example, for $t = 20$, $N = 10$, $m = 10$ and $n = 100$, the communication complexity of DPFSS is about $32n^2 + 9088n$ bytes, which ≈ 1.166 MB.

VI. CONCLUSION

In this work, we define a new model of DPSS, which is called DPFSS and allows the old committee to pass a function of shared data to a new committee. By applying Shamir's secret sharing and random coupled sharing, we design a DPFSS scheme that can transfer quadratic functions of the shared data. In the proposed construction, we also use Feldman's scheme to commit to secret sharings, in order to make the correctness of the shares verifiable. Our construction satisfies the properties of integrity and secrecy, and acheieves a communication complexity of $\mathcal{O}(n^2)$. We implement the proposed construction and show its efficiency with experimental results. It is an interesting open problem to construct DPFSS schemes for polynomials of degree > 2.

REFERENCES

[1] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[2] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. AFIPS Nat. Comput. Conf.*, 1979, pp. 313–317.
[3] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electron. Commun. Jpn. III, Fundam. Electron. Sci.*, vol. 72, no. 9, pp. 56–64, Jan. 1989.
[4] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein, "Optimized honest-majority MPC for malicious adversaries—Breaking the 1 billion-gate per second barrier," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 843–862.
[5] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proc. CRYPTO*, 1987, pp. 11–19.
[6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.
[7] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Proc. CRYPTO*, 1990, pp. 307–315.
[8] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. EUROCRYPT*, 2005, pp. 457–473.
[9] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," in *Proc. PODC*, 1991, pp. 51–59.
[10] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Proc. CRYPTO*, 1995, pp. 339–352.

[11] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung, "Proactive RSA," in *Proc. CRYPTO*, 1997, pp. 440–454.

[12] T. Rabin, "A simplified approach to threshold and proactive RSA," in *Proc. CRYPTO*, 1998, pp. 89–104.

[13] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie–Hellman-group signature scheme," in *Proc. PKC*, 2003, pp. 31–46.

[14] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung, "Optimal-resilience proactive public-key cryptosystems," in *Proc. FOCS*, Oct. 1997, pp. 384–393.

[15] Y. Frankel, P. D. MacKenzie, and M. Yung, "Adaptive security for the additive-sharing based proactive RSA," in *Proc. PKC*, 2001, pp. 240–263.

[16] Y. Desmedt and S. Jajodia, "Redistributing secret shares to new access structures and its applications," George Mason Univ., Fairfax, VA, USA, Tech. Rep. ISSE-TR-97-01, 1997.

[17] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "Storing and retrieving secrets on a blockchain," in *Proc. PKC*, 2022, pp. 1–60.

[18] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "CHURP: Dynamic-committee proactive secret sharing," in *Proc. CCS*, Nov. 2019, pp. 2369–2386.

[19] T. M. Wong, C. Wang, and J. M. Wing, "Verifiable secret redistribution for archive systems," in *Proc. 1st Int. IEEE Secur. Storage Workshop*, Dec. 2002, pp. 94–105.

[20] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. FOCS*, 1987, pp. 427–438.

[21] J. Baron, K. E. Defrawy, J. Lampkins, and R. Ostrovsky, "Communication-optimal proactive secret sharing for dynamic groups," in *Proc. ACNS*, 2015, pp. 23–41.

[22] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," U.S. Patent 4 633 470, Dec. 1986. [Online]. Available: https://www.freepatentsonline.com/4633470.html

[23] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Proc. ASIACRYPT*, Dec. 2010, pp. 177–194.

[24] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, "COBRA: Dynamic proactive secret sharing for confidential BFT services," in *Proc. IEEE Symp. Security Privacy (SP)*, 2022, pp. 1335–1353.

[25] D. Schultz, B. Liskov, and M. Liskov, "MPSS: Mobile proactive secret sharing," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, pp. 1–32, Dec. 2010.

[26] L. Zhou, F. B. Schneider, and R. Van Renesse, "APSS: Proactive secret sharing in asynchronous systems," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 3, pp. 259–286, Aug. 2005.

[27] B. Hu, Z. Zhang, H. Chen, Y. Zhou, H. Jiang, and J. Liu, "DYCAPS: Asynchronous proactive secret sharing for dynamic committees," *IACR Cryptol. ePrint Arch.*, vol. 2022, no. 1169, Jan. 2022. [Online]. Available: https://eprint.iacr.org/2022/1169

[28] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, "Long live the honey badger: Robust asynchronous DPSS and its applications," in *Proc. 32nd USENIX Secur. Symp. (USENIX Security)*, Aug. 2023, pp. 5413–5430.

[29] J. Zhang, K. Kim, H. J. Kim, D. Meyer, W. Park, S. A. Lee, Y. Dai, B. Kim, H. Moon, J. V. Shah, K. E. Harris, B. Collar, K. Liu, P. Irazoqui, H. Lee, S. A. Park, P. S. Kollbaum, B. W. Boudouris, and C. H. Lee, "Smart soft contact lenses for continuous 24-hour monitoring of intraocular pressure in glaucoma care," *Nature Commun.*, vol. 13, no. 5518, pp. 1–15, Sep. 2022.

[30] R. Bhan, R. Pamula, P. Faruki, and J. Gajrani, "Blockchain-enabled secure and efficient data sharing scheme for trust management in healthcare smartphone network," *J. Supercomput.*, vol. 79, no. 14, pp. 16233–16274, Apr. 2023.

[31] S. Singh, S. K. Sharma, and P. Mehrotra, "Blockchain technology for efficient data management in healthcare system: Opportunity, challenges and future perspectives," *Materials Today, Proc.*, vol. 62, no. 7, pp. 5042–5046, 2022.

[32] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, "Integrating blockchain for data sharing and collaboration in mobile healthcare applications," in *Proc. PIMRC*, 2017, pp. 1–5.

[33] M. Jawurek, M. Johns, and F. Kerschbaum, "Plug-in privacy for smart metering billing," in *Proc. PETS*, 2011, pp. 192–210.

[34] S. S. M. Chow, M. Li, Y. Zhao, and W. Jin, "Sipster: Settling IOU privately and quickly with smart meters," in *Proc. ACSAC*, Dec. 2021, pp. 219–234.

[35] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *Proc. FOCS*, 1985, pp. 383–395.

[36] T. M. Wong, C. Wang, and J. M. Wing, "Verifiable secret redistribution for threshold sharing schemes," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-02-114-R, 2002.

[37] M. Fitzi, J. Garay, S. Gollakota, C. P. Rangan, and K. Srinathan, "Round-optimal and efficient verifiable secret sharing," in *Proc. TCC*, 2006, pp. 329–342.

[38] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. STOC*, 1988, pp. 1–10.

[39] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Proc. CRYPTO*, 2007, pp. 572–590.

[40] M. Yoshida and S. Obana, "Verifiably multiplicative secret sharing," in *Proc. ICITS*, 2017, pp. 73–82.

[41] M. Yoshida and S. Obana, "Compact verifiably multiplicative secret sharing," in *Proc. Int. Symp. Inf. Theory Its Appl. (ISITA)*, Oct. 2020, pp. 437–441.

[42] E. Boyle, N. Gilboa, and Y. Ishai, "Breaking the circuit size barrier for secure computation under DDH," in *Proc. CRYPTO*, 2016, pp. 509–539.

[43] X. Chen, L. F. Zhang, and J. Liu, "Verifiable homomorphic secret sharing for low degree polynomials," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 4, pp. 2882–2895, Jul. 2022.

[44] G. Tsaloli, B. Liang, and A. Mitrokotsa, "Verifiable homomorphic secret sharing," in *Proc. ProvSec*, 2018, pp. 40–55.

[45] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Proc. EUROCRYPT*, vol. 2, 2015, pp. 337–367.

[46] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proc. CCS*, Oct. 2016, pp. 1292–1303.

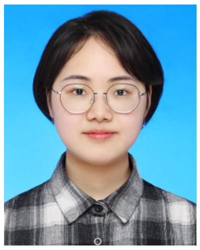
[47] J. Luo, L. F. Zhang, F. Lin, and C. Lin, "Efficient threshold function secret sharing with information-theoretic security," *IEEE Access*, vol. 8, pp. 6523–6532, 2020.

[48] J. Shen, X. Sun, F. Wei, and D. Liu, "Efficient cloud-aided verifiable secret sharing scheme with batch verification for smart cities," *Future Generat. Comput. Syst.*, vol. 10, pp. 450–456, Aug. 2020.

[49] T.-X. He and P. Shiue, "A note on horner's method," *J. Concrete Applicable Math.*, vol. 10, pp. 1–12, Jan. 2012.

[50] F. Johansson. (2018). *Flint: Fast Library for Number Theory*. [Online]. Available: https://flintlib.org/doc/index.html

[51] D. F. Aranha. (2013). *Relic is an Efficient Library for Cryptography*. [Online]. Available: http://code.google.com/p/relic-toolkit/



HONG CHEN received the bachelor's degree in information and computing science from Huazhong Agricultural University, Hubei, China. She is currently pursuing the master's degree with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China.

LIANGFENG ZHANG received the Ph.D. degree in cryptography and information security from Nanyang Technological University, Singapore. Currently, he is an Associate Professor with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China. His current research interests include cryptography and coding theory.