Storing and Retrieving Secrets on a Blockchain

Vipul Goyal Abhiram Kothapalli Elisaweta Masserova Bryan Parno Yifan Song
Carnegie Mellon University

April 30, 2020

Abstract

Multiple protocols implementing exciting cryptographic functionalities using blockchains such as time-lock encryption, one-time programs and fair multi-party computation assume the existence of a cryptographic primitive called *extractable witness encryption.* Unfortunately, there are no known efficient constructions (or even constructions based on any well studied assumptions) of extractable witness encryption. In this work, we propose a protocol that uses a blockchain itself to provide a functionality that is effectively the same as extractable witness encryption. By making small adjustments to the blockchain code, it is possible to easily implement applications that rely on extractable witness encryption and existed only as theoretical designs until now. There is also potential for new applications. As a key building block, our protocol uses a new and highly efficient batched *dynamic proactive secret sharing* scheme which may be of independent interest. We provide a proof-of-concept implementation of the extractable witness encryption construction and the underlying dynamic proactive secret sharing protocol.

1 Introduction

In the last few years numerous blockchain-based constructions for exciting and previously considered impossible cryptographic primitives have emerged. Liu et al. [LJKW18] propose a time-lock encryption scheme that allows one to encrypt a message such that it can only be decrypted once a certain deadline has passed, without relying on trusted third parties or imposing a high computational overhead on the receiver. The construction of Choudhuri et al. [CGJ$^+$17] achieves fairness in multi-party computation against a dishonest majority, which is a problem with known impossibility results in the standard model. Goyal and Goyal [GG17] present a first construction for one-time programs (that run only once and then "self-destruct") that does not use tamper-proof hardware. These constructions have something in common - they all rely on blockchains and the notion of extractable *witness encryption.* Indeed, it almost seems like blockchains and witness encryption is a "match made in heaven". Introduced by Garg et al. [GGSW13], roughly, a witness encryption scheme is a primitive that allows one to encrypt a message with respect to a particular problem instance. Such problem instance could be a sudoku puzzle in a newspaper or an allegedly bug-free program. In general, it can be any NP search problem. If the decryptor knows a valid witness for the corresponding problem instance, such as a sudoku solution or a bug in the program, he can decrypt the ciphertext. *Extractable* security is a strong notion of security for witness encryption. If a witness encryption scheme is extractable, then an adversary who is able to learn some non-trivial information about the encrypted message is also able to provide a witness for the corresponding problem instance. Unfortunately, extractable witness encryption usually requires a strong form of obfuscation, so-called *differing-inputs* obfuscation [BGI$^+$12, ABG$^+$13]. Currently, there are no practical extractable witness encryption schemes known. In fact, there exist now constructions based on standard assumptions and the work of Garg et al. [GGHW14] suggests that it even might be impossible to achieve extractable witness encryption with arbitrary auxiliary inputs.

In this work, we use a blockchain to achieve a functionality that is essentially equivalent to extractable witness encryption. Roughly, we allow users to encode a secret and specify a release condition for this secret. A predefined set of blockchain users (in the following, *miners*) will then jointly and securely store the encoding and release the secret once the release condition is satisfied. We introduce a formal definition for extractable witness encryption on blockchain (*eWEB*) and a protocol that can augment existing blockchains with this functionality. By making only small changes to the miner code, it is now possible to easily implement all the applications that use extractable witness encryption as a building block (see §5). Conveniently, many such constructions ([LJKW18, CGJ$^+$17, GG17]) already rely on the guarantees provided by a blockchain, so building extractable witness encryption into the blockchain does not change the assumptions made by these protocols. We provide a formal proof of security of our construction, relying on the guarantees provided by the blockchain setting. Specifically, we select a set (or rather, a multiset) of miners such that the majority of the selected miners are honest. As pointed out by Goyal and Goyal [GG17], one way to select such set of miners is by selecting miners who were responsible for mining the last n blocks (where n is large enough). Indeed, a single miner might mine multiple blocks and if so, appears multiple times in the set. For proof of work blockchains such as Bitcoin, the probability of successful mining is proportional to the amount of computational power, and hence if a majority of the computing power is honest, majority of the selected miners can be expected to be honest. For proof of stake blockchains, where the probability of successful mining is proportional to the amount of coins possessed by the miner, this property follows from the assumption that honest miners possess the majority of stake in the system.

Scheme	**Dynamic setting**	**Adversary**	**Threshold**	**Network**	**Comm. (amort.)**	**Comm. (non-amort.)**
Herzberg et al. [HJKY95]	No	Active	$t/n < 1/2$	synch.	$O(n^2)$	$O(n^2)$
Cachin et al. [CKLS02]	No	Active	$t/n < 1/3$	asynch.	$O(n^4)$	$O(n^4)$
Desmedt et al. [DJ97]	Yes	Passive	$t/n < 1/3$	asynch.	$O(n^2)$	$O(n^2)$
Wong et al. [WWW02]	Yes	Active	$t/n < 1/2$	synch.	$exp(n)$	$exp(n)$
Zhou et al. [ZSVR05]	Yes	Active	$t/n < 1/3$	asynch.	$exp(n)$	$exp(n)$
Schultz-MPSS [SLL08]	Yes	Active	$t/n < 1/3$	asynch.	$O(n^4)$	$O(n^4)$
Baron et al. [BDLO15]	Yes	Active	$t/n < 1/2 - \epsilon$	synch.	$O(1)$	$O(n^3)$
CHURP [MZW$^+$19]	Yes	Active	$t/n < 1/2$	synch.	$O(n^2)$	$O(n^2)$
This work	Yes	Active	$t/n < 1/2$	synch.	$O(n)$	$O(n^2)$

Figure 1: Comparison of PSS Schemes. The Comm. columns indicate the communication cost per secret in each hand-off round.

To ensure that dishonest miners cannot leak the user's secret, our extractable witness encryption scheme is built on top of a *secret sharing* scheme. A secret sharing scheme enables one party to distribute shares of a secret to n parties and ensures that an adversary in control of t out of n parties will learn no information about the secret. In our protocol, the miners hold the shares of the secret. Since we assume that the majority of the selected miners are honest, using a threshold of $t = \frac{n}{2} - 1$ enables us to securely store the secret. However, traditional secret sharing schemes are insufficient for our setting. In our extractable witness encryption construction, the parties holding the shares of the secrets are the miners. These can join or leave the system at any time. Thus, the set of parties holding the secret shares is constantly changing. To achieve security in this case, *dynamic proactive secret sharing* (DPSS) is required ([HJKY95, DJ97, SLL08]). A part of our protocol is a new and highly optimized batched DPSS scheme. We are specifically interested in the batched setting as there might be thousands of secrets stored in the system at any given time and we need an efficient way to update all of those secrets in parallel. In contrast to previous work on batched DPSS [BDLO15] that focused on a single client submitting a batch of secrets and does not allow to store and release secrets independently, we allow for multiple different clients dynamically sharing and releasing secrets. Our construction is the most efficient DPSS scheme that allows the highest-possible adversarial threshold of $\frac{1}{2}$. We have formally proven secure and implemented this new scheme and believe that it is of independent interest (see Figure 1 for a comparison with other PSS schemes).

In summary, we make the following contributions:

- We propose a new cryptographic primitive - extractable witness encryption on blockchain (§3).
- We design and formally prove secure a protocol which satisfies the notion of extractable witness encryption on blockchain (§4).
- We present and formally prove secure a highly efficient batched dynamic proactive secret sharing construction (§2).

Organization We begin by discussing DPSS - the key building block in our system (§2). We discuss the DPSS functionality, including adversarial model and definition of security, and provide an overview of our DPSS construction. We provide the full construction in Appendix C. Next, we formally define extractable witness encryption on blockchain in Section 3, including syntax (§3.1), and security definition (§3.2). In Section 4.2 we provide our construction for extractable witness encryption on blockchain, and we give its security proof in Appendix E. In Section 5 we discuss various applications that can be implemented using our constructions. We discuss related work on dynamic proactive secret sharing and extractable witness encryption in Section 6.

2 Dynamic Proactive Secret Sharing (DPSS)

We start by discussing dynamic proactive secret sharing, which is the key building block of our extractable witness encryption on blockchain construction. We first informally explain the DPSS process, and provide the adversarial model and the formal definition of its security properties. Then, we give an overview of our protocol. The full construction and its security proof are provided in the Appendix (C and D).

2.1 DPSS Background

A dynamic proactive secret sharing scheme (DPSS) allows a party to distribute shares of a secret to n parties. The scheme ensures that an adversary in control of some threshold number of parties t will learn no information about the secret. Over the course of running the protocol the set of parties holding the secret is constantly changing, and the adversary might "release" some parties (corresponding users regain control of their systems) and corrupt new ones.

A DPSS scheme typically consists of three phases: setup, hand-off, and reconstruction.

In the setup phase, one or more independent clients secret-share a total of m secrets to a set of n parties, known as a committee, denoted by $\mathcal{C} = \{P_1, \ldots, P_n\}$. After the setup phase, each committee member holds one share for each secret s.

As the protocol runs, the hand-off phase is periodically invoked. In the hand-off phase, the sharing of each secret is passed from the old committee, $\mathcal{C}$, to a new committee, $\mathcal{C}'$. This process reflects parties leaving and joining the committee. After the hand-off phase, all parties in the old committee delete their shares, and all parties in the new committee hold a sharing for each secret s. The hand-off phase is particularly challenging, since during the hand-off a total of $2t$ *parties* may be corrupted (t parties in the old committee and t parties in the new committee).

When a client (which need not be one of the clients that secret-shared the secrets in the setup phase) asks for the reconstruction of some specific secret $s^\star$, all parties in the current committee and the client engage in a reconstruction process to allow the client learn the secret $s^\star$.

2.1.1 Adversary Model

We consider a fully malicious adversary $\mathcal{A}$ with the power to adaptively choose parties to corrupt at any time. $\mathcal{A}$ can corrupt any number of clients in the setup phase and learn the secrets held by the corrupted clients. When a party P_i is corrupted by $\mathcal{A}$, $\mathcal{A}$ can arbitrarily control the behavior of P_i and modify the memory state of P_i. Even if $\mathcal{A}$ releases its control of some party P_i, the memory state of P_i may have already been modified; e.g., its share might be erased.

Note that for a party P_i in both the old committee $\mathcal{C}$ and the new committee $\mathcal{C}'$, if $\mathcal{A}$ has the control of P_i during the hand-off phase, then P_i is considered to be corrupted in both committees. If $\mathcal{A}$ releases its control before the hand-off phase in which the secret sharing is passed from $\mathcal{C}$ to $\mathcal{C}'$, then P_i is only considered to be corrupted in the old committee $\mathcal{C}$. Similarly, if $\mathcal{A}$ only corrupts P_i after the hand-off phase, P_i is only considered to be corrupted in the new committee $\mathcal{C}'$.

In this paper, we focus on *computationally bounded* adversaries. For each committee $\mathcal{C}$ with a threshold $t < |\mathcal{C}|/2$, $\mathcal{A}$ can corrupt at most t parties in $\mathcal{C}$.

For simplicity, in the following, we assume that there exist secure point-to-point channels between the parties and the corruption threshold is a fixed value t. Our construction can be easily adapted to allow different thresholds for different committees (see Appendix C.5 for more details).

2.1.2 DPSS Security Definition

A dynamic proactive secret-sharing scheme is required to satisfy two properties, *robustness* and *secrecy*. At a high-level, *robustness* requires that it should always be possible to recover the secret. *Secrecy* requires that an adversary should not learn any further information about the secret beyond what has been learned before running the protocol. Our formal definition is similar to the one used by Baron et al. [BDLO15], but in contrast to their work we consider not only the scenario of one client submitting the secrets, but many different clients submitting (and requesting release of) secrets at different points in time.

Robustness. For any PPT adversary $\mathcal{A}$ with corruption threshold t holds that after each setup phase, there exists a fixed secret $s^\star$ for the sharing distributed during this phase. If the setup phase was executed by an honest client, this secret is the same as the one chosen by this client. When at some point an honest client asks for a reconstruction of this secret, the client receives the correct secret $s^\star$.

Secrecy. For any PPT adversary $\mathcal{A}$ with corruption threshold t, there exists a simulator $\mathcal{S}$ with access to an ideal $\mathsf{Ideal}_{\text{safe}}$ (described in Procedure 2), such that the view of $\mathcal{A}$ interacting with $\mathcal{S}$ is computationally indistinguishable from the view in the real execution.

Ideal Procedure 2: $\mathsf{Ideal}_{\text{safe}}$

1. $\mathsf{Ideal}_{\text{safe}}$ receives the secrets from the clients in the setup phase.
2. When an honest client asks for the reconstruction of some specific secret $s^\star$, $\mathsf{Ideal}_{\text{safe}}$ sends $s^\star$ to that client.

Definition 2.1. A dynamic proactive secret-sharing scheme is secure if for any PPT adversary $\mathcal{A}$ and threshold t, it satisfies both **robustness** and **secrecy**.

2.2 Overview: Our DPSS Construction

We now provide an overview of our batched DPSS construction. We first discuss the hand-off phase of our construction in the semi-honest case (§2.2.1) and then explain how it can be modified when the adversary is fully malicious (§2.2.2). Finally, we elaborate on the setup (§2.2.3) and reconstruction phases (§2.2.4), and provide an intuition for the security proof of our construction (§2.2.5).

In the following, we assume the corruption threshold for each committee is fixed to t. The construction is based on Shamir Secret Sharing Scheme [Sha79]. We use $[x]_d$ to denote a degree-d sharing, i.e., $(d+1)$-out-of-n Shamir sharing. It requires at least $d+1$ shares to reconstruct the secret and any d or fewer shares do not leak any information about the secret. Note that Shamir Secret Sharing is additively homomorphic.

2.2.1 Skeleton of Our Construction: Semi-honest Case

We first explain the high-level idea of our protocol in the semi-honest setting; i.e., all parties honestly follow the protocol. The crux of our construction is that both the old and the new committee hold a sharing of a random value. While the *sharing* is different for the two committees, the *value* this sharing corresponds to is the same. Let $([r]_t, [\tilde{r}]_t)$ denote these two sharings, where $[r]_t$ is held by the old committee, $[\tilde{r}]_t$ is held by the new committee, and $r = \tilde{r}$. Suppose the secret sharing we want to refresh is $[s]_t$, held by the old committee. Then the old committee will compute the sharing $[s+r]_t = [s]_t + [r]_t$ and reconstruct the secret $s+r$. Since r is a uniform element, $s+r$ does not leak any information about s. Now, the new committee can compute $[\tilde{s}]_t = (s+r) - [\tilde{r}]_t$. Since $\tilde{r} = r$, we have $\tilde{s} = s$. This whole process is split into *preparation* and *refresh* phases:

- In the preparation phase, parties in the new committee prepare two degree-t sharings of the same random value $r(=\tilde{r})$, denoted by $[r]_t$ and $[\tilde{r}]_t$. The old committee receives the shares of $[r]_t$ and the new committee holds the shares of $[\tilde{r}]_t$. We refer to these two sharings as a *coupled sharing.*

- In the refresh phase, the old committee reconstructs the sharing $[s]_t + [r]_t$ and publishes the result. The new committee sets $[\tilde{s}]_t = (s + r) - [\tilde{r}]_t$.

We start by explaining the *preparation phase* with the goal of generating a coupled sharing of a random value. In the following, let $\mathcal{C}$ denote the old committee and $\mathcal{C}'$ denote the new committee. Intuitively, the new committee can prepare a coupled sharing as follows:

1. Each party $P'_i \in \mathcal{C}'$ prepares a coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ of a random value and distributes $[u^{(i)}]_t$ to the old committee and $[\tilde{u}^{(i)}]_t$ to the new committee.

2. All parties in the old committee compute $[r]_t = \sum_{i=1}^{n} [u^{(i)}]_t$. All parties in the new committee compute $[\tilde{r}] = \sum_{i=1}^{n} [\tilde{u}^{(i)}]_t$.

Since for each i holds $u^{(i)} = \tilde{u}^{(i)}$, we have $r = \tilde{r}$. However, this way of preparing coupled sharings is wasteful since at least $(n-t)$ coupled sharings are generated by honest parties, which are uniformly random to corrupted parties. In order to get $(n-t)$ random coupled sharings instead of just 1, we borrow an idea from Damgård et al. [DN07].

In the work of Damgård et al., parties need to prepare a batch of random sharings which will be used in an MPC protocol. All parties first agree on a fixed and public Vandermonde matrix $\boldsymbol{M}^{\mathrm{T}}$ of size $n \times (n-t)$. An important property of a Vandermonde matrix is that any $(n-t) \times (n-t)$ submatrix of $\boldsymbol{M}^{\mathrm{T}}$ is *invertible.* To prepare a batch of random sharings, each party P_i generates and distributes a random sharing $[u^{(i)}]_t$. Next, all parties compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}},$$

and take $[r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t$ as output. Since any $(n-t) \times (n-t)$ submatrix of $\boldsymbol{M}$ is invertible, given the sharings distributed by corrupted parties, there is a one-to-one map from the output sharings to the sharings distributed by honest parties. If the input sharings of the honest parties are uniformly random, the one-to-one mapping ensures that the output sharings are uniformly random as well [DN07].

Note that any linear combination of a set of coupled sharings is also a valid coupled sharing. Thus, in our protocol, instead of computing $([r]_t, [\tilde{r}]_t) = \sum_{i=1}^{n}([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$, parties in the old committee can compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}},$$

and parties in the new committee can compute

$$([\tilde{r}^{(1)}]_t, [\tilde{r}^{(2)}]_t, \ldots, [\tilde{r}^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([\tilde{u}^{(1)}]_t, [\tilde{u}^{(2)}]_t, \ldots, [\tilde{u}^{(n)}]_t)^{\mathrm{T}}.$$

Now all parties get $(n-t)$ random coupled sharings. The amortized cost of communication per pair is $O(n)$ elements.

We now describe the *refresh phase.* For each sharing $[s]_t$ which needs to be refreshed, one random coupled sharing $([r]_t, [\tilde{r}]_t)$ is consumed. Parties in the old committee first select a special party $P_{\mathtt{king}}$. To reconstruct $[s]_t + [r]_t$, parties in the old committee locally compute their shares of $[s]_t + [r]_t$, and then send the shares to $P_{\mathtt{king}}$. Then, $P_{\mathtt{king}}$ uses these shares to reconstruct $s + r$ and publishes the result. Finally, parties in the new committee can compute $[\tilde{s}]_t = (s + r) - [\tilde{r}]_t$.

2.2.2 Moving to a Fully-Malicious Setting

When faced with a fully-malicious adversary, three problems arise.

- In the preparation phase, a corrupted party may distribute an inconsistent degree-t sharing or incorrect coupled sharing.
- In the refresh phase, a corrupted party may provide incorrect share to $P_{\texttt{king}}$, which results in a reconstruction failure.
- A malicious $P_{\texttt{king}}$ may provide an incorrectly reconstructed value.

We address these problems by checking the correctness of couple sharings in the preparation phase and relying on polynomial commitments to transform a plain Shamir secret sharing into a verifiable one.

Checking the Correctness of Coupled Sharings. Recall that any linear combination of coupled sharings is also a valid coupled sharing. Thus, to increase efficiency, instead of checking the correctness of *each* coupled sharing, it is possible to check *a random linear combination* of the coupled sharings distributed by each party.

To protect the privacy of the coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ generated by P_i' , P_i' will generate one additional random coupled sharing as a random mask, which is denoted by $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t)$.

Consider the following two sharings of polynomials of degree-$(2n-1)$:

$$\begin{aligned}
[F(X)]_t &= \sum_{i=1}^{n}([\mu^{(i)}]_t + [u^{(i)}]_t \cdot X)X^{2(i-1)}, \\
[\tilde{F}(X)]_t &= \sum_{i=1}^{n}([\tilde{\mu}^{(i)}]_t + [\tilde{u}^{(i)}]_t \cdot X)X^{2(i-1)}.
\end{aligned}$$

If all coupled sharings are correct, then $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ is also a coupled sharing for any λ. Otherwise, the number of λ such that $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ is a coupled sharing is bounded by $2n-1$. Therefore, it is sufficient to test $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ at a random evaluation point λ. Note that each individual coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ is masked by $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t)$. Therefore, revealing $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ does not leak any information about the individual coupled sharings $\{([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)\}_{i=1}^{n}$.

Therefore, all parties first generate a random challenge λ (see Appendix C.1). Parties in the old committee compute $[F(\lambda)]_t$ and then publish their shares. Parties in the new committee compute $[\tilde{F}(\lambda)]_t$ and then publish their shares. Finally, all parties check whether $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ is a valid coupled sharing.

If the check fails, we need to pinpoint the parties who distributed incorrect coupled sharings. Each coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ is masked by $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t)$. Therefore it is safe to open the whole sharing $([\mu^{(i)}]_t + [u^{(i)}]_t \cdot \lambda, [\tilde{\mu}^{(i)}]_t + [\tilde{u}^{(i)}]_t \cdot \lambda)$ and check whether it is a valid coupled sharing. For each i, parties in the old committee compute $[\mu^{(i)}]_t + [u^{(i)}]_t \cdot \lambda$ and publish their shares, and parties in the new committee compute $[\tilde{\mu}^{(i)}]_t + [\tilde{u}^{(i)}]_t \cdot \lambda$ and publish their shares. This way, we can tell which coupled sharings are inconsistent. This inconsistency has two possible causes:

- The coupled sharing distributed by a party (in the following, *dealer*) were invalid (secrets were not the same or t-sharing was invalid).

- A corrupted party provided an incorrect share.

The first case implies that the dealer is a corrupted party. To rule out the second possibility, we will rely on polynomial commitments, which can be used to transform a plain Shamir secret sharing into a verifiable one so that an incorrect share can be identified and rejected by all parties.

Relying on Polynomial Commitments. A degree-t Shamir secret sharing corresponds to a degree-t polynomial $f(\cdot)$ such that: **(a)** the secret is $f(0)$, and **(b)** the i-th share is $f(i)$. Thus, each dealer can commit to f by using a polynomial commitment scheme (A.5) to add verifiability.

A polynomial commitment scheme allows the dealer to open one evaluation of f (which corresponds to one share of the Shamir secret sharing) and the receiver can verify the correctness of this evaluation value. Essentially, whenever a dealer distributes a share it also provides a *witness* which can be used to verify this share. Informally, a polynomial commitment scheme should satisfy two binding properties and one hiding property:

- Polynomial Binding: One commitment cannot be opened to two different polynomials.
- Evaluation Binding: One commitment cannot be opened to two different values at the same evaluation point.
- Hiding: The commitment should not leak any information about the committed polynomial.

We use the polynomial commitment as follows: In the beginning, each dealer first commits to the sharings it generated and opens the shares to corresponding parties. To ensure that each party is satisfied with the shares it received, there is an accusation-and-response phase that proceeds as follows:

1. Each party publishes $(\texttt{accuse}, P'_i)$ if the share received from P'_i does not pass the KZG verification algorithm.
2. For each accusation made by P_j, P'_i opens the j-th share to all parties, and P_j uses the new share published by P'_i.

Note that an honest party will never accuse another honest party. Additionally, if a malicious party accuses an honest party, no more information is revealed to the adversary than what the adversary knew already. Therefore it is safe to reveal the share sent from P'_i to P_j. *After this step, all parties should always be able to provide valid witnesses for their shares.*

From now on, each time a party sends or publishes a share, this party also provides *the associated witness* to allow other parties verify the correctness of the share.[1] Since honest parties will always provide shares with valid witnesses and there are at least $n - t \geq t + 1$ honest parties, all parties will only use shares that pass verification. Intuitively, this solves the problem of incorrect shares provided by corrupted parties since corrupted parties cannot provide valid witnesses for those shares. Due to a subtle limitation of the KZG commitment scheme, actually we need to add an additional minor verification step (see Appendix B for details).

We give the complete description of the hand-off protocol in Figure 18 in Appendix C.1.

[1]Sometimes parties will need to compute linear operations on sharings. Since in our construction we use the KZG commitment scheme [KZG10], which is linearly homomorphic, all parties can locally compute the commitment of the result.

2.2.3 DPSS Setup Phase

At a high level, the setup phase uses a similar approach to the hand-off phase. First, the committee prepares random coupled sharings. For each secret s distributed by a client, one random sharing $[r]_t$ is consumed. The client receives the whole sharing $[r]_t$ from the committee and reconstructs the value r. Finally, the client publishes $s+r$. The committee then computes $[s]_t = s+r-[r]_t$. As in the hand-off phase, the validity of the distributed shares is verified using the KZG commitment scheme. The full description of the setup phase is in Figure 25 in Appendix C.2.

2.2.4 DPSS Reconstruction Phase

When a client asks for the reconstruction of some secret $s^\star$, all parties in the current committee simply send their shares of $[s^\star]_t$ and the associated witnesses to the client. Then the client can reconstruct the secret using the first $t+1$ shares that pass the verification. The complete description of the reconstruction protocol is in Figure 26 in Appendix C.3.

2.2.5 Robustness and Secrecy of Our Construction.

We give a high-level idea of our proof.

To show robustness, note that all parties together verify the correctness of the sharings and commitments dealt by each party. This step ensures that the sharings determined by the shares held by honest parties are correct openings of the commitments. If a corrupted party provides an incorrect share with a valid witness, it effectively breaks the evaluation binding property of the KZG commitment scheme. Therefore, with overwhelming probability, incorrect shares provided by corrupted parties will be rejected, and each sharing can be reconstructed by using the shares with valid witnesses.

As for secrecy, note that for each sharing, corrupted parties receive at most t shares, which are independent of the secret. Therefore, when an honest party needs to distribute a random sharing, the simulator can send random elements to corrupted parties as their shares without fixing the shares of honest parties. Since we use the perfectly hiding variant of the KZG commitment, the commitment is independent of the secret, and can be generated using the trapdoor of the KZG commitment. Furthermore, we can adaptively open t shares chosen by the adversary after the commitment is generated. This allows our protocol to be secure against adaptive corruptions.

See Appendix D for more details.

3 Defining Extractible Witness Encryption on the Blockchain

Witness encryption allows a party to encrypt a message to an instance x of an NP language. Another party can then decrypt the ciphertext using a witness that x is in the language. The traditional notion of security for witness encryption schemes introduced by Garg et al. [GGSW13] is called *soundness security* and states that if a message was encrypted to some instance x that is *not* in the language, then no adversary can learn any non-trivial information about the encrypted message. However, this security notion is often insufficient. Many constructions [LJKW18, CGJ$^+$17, GG17] consider the case where the instance x *is* in the language, and must rely on a stronger version of security for witness encryption schemes, the so-called *extractable* security. Informally, if an

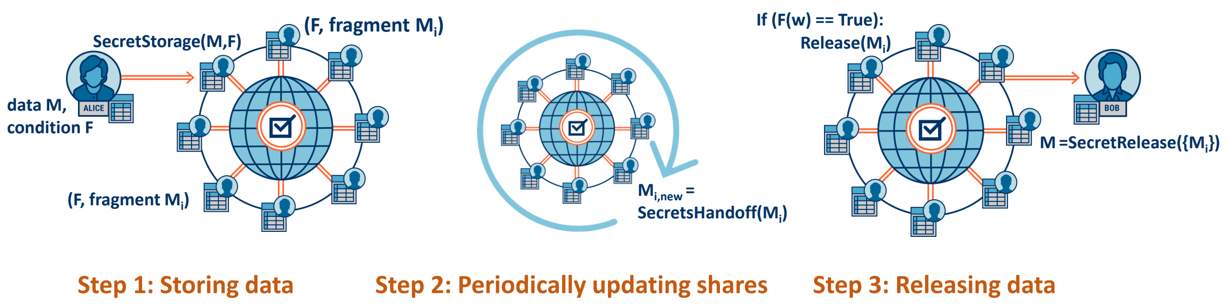

Figure 3: High-level overview of eWEB.

adversary is able to distinguish between two different ciphertexts encrypted to the same problem instance, then he is also able to provide a witness to this problem instance.

In our work we propose to use a blockchain to achieve a similar goal. In our setting we distinguish between users who deposit secrets (depositors), users who request that a secret be released (requesters), and a changing set of miners who are executing these requests. A depositor who wishes to securely store a secret until some condition is satisfied will distribute the encoded secret among the miners and specify the release condition. When a requester wishes to learn the secret, they must provide a valid witness for the release condition. The miners will check the witness, and if it is valid, provide the secret to the requester in a secure way. In addition to storing and releasing secrets, we require a hand-off procedure to be periodically executed by the miners, since the set of miners is constantly changing. During the hand-off process, the secrets are handed from the old set of miners (*old committee*) to the new set of miners (*new committee*). The full process is depicted in Figure 3. Intuitively, no adversary should learn any non-trivial information about a user's secret unless he already knows a witness for the corresponding release condition. Additionally, no one should be able to change stored secrets.

We now provide a formal syntax of the primitive in §3.1. Using this syntax, we define the security of extractable witness encryption on a blockchain by providing a security game in §3.2.

3.1 Syntax

The extractable witness encryption on blockchain (*eWEB*) system consists of the following, possibly randomized and interactive, subroutines:

$SecretStorage(M, F) \rightarrow (id, \{frag_0, .., frag_n\}, F)$: Depositor uses the SecretStorage function to store a secret M which can be released to a requester who knows a witness w s.t. $F(w)$ is true. After interacting with the depositor, each of the n miners obtains some data (in the following, we call it a "fragment" of the secret) $frag_i$ that is associated with the secret storage request with the identifier id.

$SecretsHandoff(\{frag^0_0, .., frag^0_n\}, .., \{frag^m_0, .., frag^m_n\}) \rightarrow (\{\widetilde{frag}^0_0, .., \widetilde{frag}^0_n\}, .., \{\widetilde{frag}^m_0, .., \widetilde{frag}^m_n\})$: Miners periodically execute the SecretsHandoff function to hand over all m stored secrets from the old committee to the new committee. Each miner i of the old committee possesses m fragments $frag^0_i, frag^1_i, .., frag^m_i$ at the start of the hand-off protocol. Each miner i of the new committee possesses m fragments $\widetilde{frag}^0_i, \widetilde{frag}^1_i, .., \widetilde{frag}^m_i$ at the end of the hand-off protocol.

$SecretRelease(id, w) \rightarrow M$ or $\perp$: A requester uses this function to request the release of the secret with the identifier id. The requester specifies the witness w to the decryption condition.

Miners then check whether the requester holds a valid witness. If so, as a result of the interaction with the miners, the requester obtains the secret M. Else, the function returns $\perp$, indicating a failed attempt.

3.2 Security Game Definition

We define security via a game.

Definition 3.1 (Security Game). The game is played between the adversary, $\mathcal{A}$, and a challenger, $\mathcal{C}$. $\mathcal{A}$ is a probabilistic, polynomial-time adversary who controls t parties of the old committee and t parties of the new committee during each handoff round. The game is parametrized by the number of parties n participating in each round as well as the number of rounds d. There exists a set of $t * d$ (unique) party identifiers and corresponding public keys of the honest parties. The set of public keys of honest parties is denoted by PK_H. This set is known to $\mathcal{A}$. The decryption condition F is public information as well. The game takes as input a list W of witnesses for F: $\forall w \in W: \; F(w) = true$, which is known to the challenger. Note that the same witness can occur multiple times in this list, and so if there exists at least one witness for W we assume w.l.o.g. that W is of length d. If no witness for F exists, the list is empty. Whenever messages are exchanged between parties, the adversary sees them (including messages sent from honest to honest parties).

1. $\mathcal{A}$ chooses two strings M_1 and M_2 of the same length, and submits both to the challenger.
2. The challenger flips a coin, $b \in \{0, 1\}$, uniformly at random, which is fixed for the duration of the game.
3. In each round, until $\mathcal{A}$ finishes the game, the following happens:
   The new committee is chosen and secrets (if any exist) are handed over
   - $\mathcal{A}$ chooses a set T of t indices for the adversarial parties of the new committee and generates a public- and secret key pair (pk_i, sk_i) for each adversarial member C_i of this committee. Then, $\mathcal{A}$ sends T and the public keys $\{pk_i\}_{i \in T}$ to $\mathcal{C}$.
   - $\mathcal{A}$ chooses a set H of $n - t$ public keys from the set PK_H for the honest parties of the new committee and sends H to $\mathcal{C}$.
   - If at least one secret is stored on the blockchain, $\mathcal{C}$ carries out $SecretsHandoff$ for each honest member of the old and the new committee.
4. Additionally, until $\mathcal{A}$ finishes the game, one of the following can happen in each round:
   (a) **$\mathcal{A}$ may ask the challenger to create a secret storage request for the challenge (only once)**
   - If this is the first time $\mathcal{A}$ makes such request, the challenger carries out SecretStorage for the secret M_b and the secret release condition F.

   (b) If the challenge secret storage request was executed, **$\mathcal{A}$ may ask the challenger to create a secret release request for the challenge for an honest party with the party identifier p_{id}.**
   - If p_{id} corresponds to an honest party, $\mathcal{C}$ carries out $SecretRelease$ for the challenge request using the public key pk of the party with the identifier p_{id} and a witness $w \leftarrow W$.

   (c) **$\mathcal{A}$ may create new secret deposits**
   - $\mathcal{C}$ carries out SecretStorage for the new request for each honest member of the committee using the information supplied by the adversary.

   (d) **$\mathcal{A}$ may create new release requests for any number of storage requests with IDs** id_i

- $\mathcal{C}$ carries out *SecretRelease* for each request id_i for each honest member of the committee using the information supplied by the adversary.

(e) **$\mathcal{A}$ may end the game with its guess, b', for b.**

Definition 3.2. (Secrecy) For a security parameter λ, number of parties participating in each round n, number of rounds d, corruption threshold t, decryption condition F, and list of witnesses W of size d (or 0 if no witness for F exists) s.t. $\forall w \in W : \ F(w) = true$, let the advantage $\mathsf{Adv}[\mathcal{A}, 1^\lambda, n, d, F]$ of the adversary $\mathcal{A}$ in the game introduced above be defined as follows:

$$\mathsf{Adv}[\mathcal{A}, 1^\lambda, n, d, t, F, W] = \left| Pr[b' = b] - \frac{1}{2} \right|$$

The system provides secrecy if for any polynomial-time probabilistic adversary $\mathcal{A}$ and any predicate function F that is verifiable in PPT there exists a PPT extractor $\mathcal{E}$ that uses $\mathcal{A}$ as an oracle such that:

$$\mathsf{Adv}[\mathcal{A}, 1^\lambda, n, d, t, F, W] \geq \tfrac{1}{poly(n)} \Rightarrow Pr[E^{\mathcal{A}}(F, 1^n) = w \mid F(w) = True] \geq \tfrac{1}{poly'(n)},$$

where $poly(n), poly'(n)$ are some polynomials.

Practically, this definition means that if the adversary is able to distinguish between the execution of the protocol that uses a message M_0 as a secret from the execution that uses message M_1 as a secret, then it is also possible to extract a valid witness for the decryption condition F using this adversary. Note that intuitively this notion is very similar to the extractable security of witness encryption, which states that if an adversary can distinguish between two ciphertexts then he can also extract a witness from the corresponding problem instance.

Furthermore, we define a robustness property of the scheme as follows:

Definition 3.3. (Robustness) For any PPT adversary $\mathcal{A}$ with corruption threshold t holds that after the *SecretStorage*(M, F) execution, there exists a fixed secret (identified by id) for the distributed fragmentation $\{frag_0, .., frag_n\}$. In particular, the fragmentation dealt by an honest depositor has the same secret M as the one chosen by this depositor. When at some point an honest requester executes *SecretRelease*(id, w), where $F(w) = true$, the requester reconstructs the correct secret M.

4 Our eWEB Protocol Design

In this section, we introduce our construction. First, we provide an overview of the assumptions that we rely on in our construction (§4.1). Then, we describe our eWEB construction (§4.2). Finally, we provide a security proof sketch for this construction (§4.3), with a formal proof in Appendix E.

Additionally, in Appendix F we provide a so-called PublicWitness protocol that is very similar to our eWEB construction, but targets a slightly different use case scenario. In this scenario the secret is made public after a single successful release request.

4.1 Assumptions

Adversary model. We assume that the adversary is able to control a polynomial number of users and miners, subject to the constraint that the blockchain has $(\frac{n}{2}+1, n)$-*chain quality*, meaning that for each n or more continuous blocks mined in the system, more than half were mined by honest parties. As pointed out by Goyal and Goyal [GG17], for proof of work blockchains, where the probability of successful mining is proportional to the amount of computational power, this assumption follows from the assumption that honest miners possess the majority of the computational power in the system. We assume that this majority is "significant enough" (to e.g. defeat selfish mining attacks as is required for e.g. Bitcoin security [ES14]). For proof of stake blockchains, where the probability of successful mining is proportional to the amount of coins possessed by the miner, it follows from the assumption that honest miners possess the majority of stake in the system. In practice, we pick an n that is big enough to provide this property with only a very small error probability. Our eWEB construction is compatible with both proof of work and proof of stake blockchains that satisfy this condition.

We assume that once an adversary corrupts a party it remains corrupted. The adversary cannot adaptively corrupt previously honest parties. When a party is corrupted by the adversary, the adversary can arbitrarily control this party's behaviour and internal memory state. We do not distinguish between adversarial parties and honest parties that behave maliciously unintentionally; e.g., those that have connection issues and can not access the blockchain to participate in the protocol.

Infrastructure model. It is common for public keys to be known in blockchains. In addition to this assumption, we require that each party p_i has a *unique identifier*, denoted by pid_i that is known to all other parties in the system as well. In practice, this unique identifier can be the hash of the party's public key. For simplicity, we present the protocol as if there were authenticated channels between all parties in the system. In practice, these channels can be realized using standard techniques such as signatures.

Communication model. Our DPSS scheme assumes secure point-to-point communication channels. In the decentralized blockchain setting of the ExtWe system it is preferable not to make such an assumption. As mentioned in Churp [MZW$^+$19], this assumption is undesireable since establishing direct point-to-point channels could comprise nodes' anonymity and could lead to targeted attacks. Instead, we assume that parties communicate via an existing blockchain. We distinguish between posting a message on the blockchain and using the blockchain for broadcast. The latter can be implemented e.g., using the *Transaction Ghosting* technique introduced in Churp [MZW$^+$19]. Point-to-point channels can be simulated using IND-CCA secure encryption and broadcasting the ciphertexts.

Storage. We assume that in addition to the internal storage of the parties there exists some publicly accessible off-chain storage, and it is cheaper to save a message using this storage rather than save the same message on the blockchain. The robustness of our system depends on the robustness properties of the off-chain storage and thus storage systems with a reputation for high availability should be chosen. However, as we show in Appendix E, malicious off-chain storage does not impact the secrecy guarantees of our system. Alternatively, at a higher cost, it is also possible to simply use on-chain storage for everything.

4.2 Our eWEB Construction

We now sketch the workflow of our protocol. Given a secret message M and a release condition F, the depositor stores the release condition F on the blockchain and secret-shares M among the miners using the secret storage (setup) algorithm of the DPSS scheme. During the periodically executed hand-off phase of the protocol, the secrets are passed from the miners of the old committee to the miners of the new committee (we elaborate on how these committees are chosen in §4.2.1) using the DPSS hand-off algorithm. In order to retrieve a stored secret, a requester U needs to prove that they are eligible to do so. This poses a challenge. An insecure solution is to just send a witness w to the miners such that the release condition is satisfied: $F(w) = \mathit{true}$. One obvious problem with this solution is the fact that a malicious miner is able to use the provided witness to construct a new secret release request and retrieve the secret himself. To solve this problem, instead of sending the witness in clear, the user will *prove that they know a valid witness.* Thus, while the committee members are able to check the validity of the request and privately release the secret to U, the witness remains hidden. In the protocol, we rely on non-interactive zero knowledge proofs (see 4.2.3) for relation $R = \{(pk, w) \mid F(w) = \mathit{true} \text{ and } pk = pk\}$, where $F(\cdot)$ is the secret release condition submitted by the depositor and pk is the public key of user U and is used to identify the user eligible to receive the secret. After the miners verified the validity of the request, they engage in the DPSS's secret reconstruction algorithm with the requester U to release the secret to U.

We provide the full secret storage protocol in Figure 6. The hand-off protocol is given in Figure 7. The secret release protocol is in Figure 8. The complete construction is provided in Figure 9. We now elaborate on the details we left out from the above sketch of the construction.

4.2.1 Choosing miner committee.

At any point of time, all parties must know the current committee. In our protocol, we chose the initial committee to be the miners that mined the most recent n blocks. Then, when the hand-off phase is executed, the old committee passes the secrets to the miners that mined the most recent n blocks - these constitute the new committee. Note that this way the size of the committee never changes and all parties can determine the current committee by looking at the blockchain state. It is possible that some committee miners receive more information about the secrets than others - roughly, if a party mined m out of the last n blocks, this party receives $\frac{m}{n}$ of all the shares. This reflects the distribution of the computing power (for POW blockchains) or stake (for POS blockchains) in the system.

4.2.2 Handling changing committees.

To handle the changing set of miners in the committee holding the secrets, we employ our DPSS construction (§2.2). Hereby we access all DPSS algorithms (setup, hand-off, secret reconstruction) in a black-box way.

4.2.3 Proving knowledge of the witness.

To prove the knowledge of a valid witness, we rely on non-interactive zero knowledge proofs (NIZKs). Roughly, such proofs allow one party (the prover) to prove validity of some statement to another party (the verifier), in a way that nothing except for the validity of the statement is revealed. In our construction we specifically rely on non-interactive zero knowledge *proofs of knowledge*, which allow the prover convince the verifier that they know a witness to some statement. We provide formal definitions of the NIZK's security properties we rely on in Appendix A.2.

4.2.4 Point-to-point channels/Subtleties.

As mentioned in §4.1, while our DPSS protocol assumes secure point-to-point channels, we do not make such an assumption in the witness encryption construction. Instead, we rely on authenticated encryption and Protocols 4 and 5. Protocol 4 is executed whenever a message needs to be securely sent from one party to another. Note that it is employed not only for the messages exchanged in the ExtWe protocol, but also for the messages of the underlying DPSS protocol. Whenever a party receives an encrypted message, it performs an authentication check via Protocol 5 to ensure that a ciphertext received from some party was generated by the same party. This prevents the following malleability issue - a malicious user desiring to learn a secret with the identifier id could generate a new secret storage request with a function F for which he knows a witness, copy the DPSS messages sent by the user who created the storage request id to the miners and later on prove his knowledge of a witness for F to release the corresponding secret. Without the authentication check our construction would be insecure and our security proof (see Appendix E) would not go through.

4.2.5 Handling large secrets/Using off-chain storage.

Since the secret itself might be very large, it is also possible to first encrypt the secret using a symmetric encryption scheme, store the ciphertext publicly off chain and then secret-share the key instead. Also, we store request parameters (such as release conditions or proofs) off-chain, saving only the hash of the message on-chain.

Protocol 4: MessagePreparation

1. For a message m to be sent by party P_s to party P_r, P_s computes the following ciphertext $c \leftarrow Enc_{pk_r}(m|pid_s)$, where pk_r is the public key of P_r and pid_s is the party identifier of P_s.
2. P_s prepends the storage identifier id of his request and sends the tuple (id, c) to P_r. The storage identifier is specified to let the receiving party know for which request the data in c is meant to be used.

4.3 Security Proof Intuition

We formally prove the security of our hidden witness protocol in Appendix E. Informally, we show that the hidden witness protocol satisfies the security definition for extractable witness enryption on blockchain given in §3.2 by providing a hybrid proof. In this proof, we rely on the unbounded zero-knowledge and simulation sound extractability properties of the NIZK scheme to switch from providing honest proofs to using simulated proofs. Next, we rely on the collision-resistance of the hash function to show that any modification of the data stored offchain will be detected. Then, we rely on the multi-message IND-CCA security of the encryption scheme to change all encrypted messsages exchanged between honest parties to encryptions of zero. Finally, we rely on the secrecy

Protocol 5: AUTHENTICATEDDECRYPTION

1. Upon receiving a tuple (id, c) from party P_s over an authenticated channel, the receiving party P_r decrypts c using its secret key sk to obtain $m \leftarrow Dec_{sk}(c)$.
2. P_r verifies that m is of the form $m'|pid_s$ for some message m', where pid_s is the identifier of party P_s.
3. If the verification check fails, P_r stops processing c and outputs an error message.

Protocol 6: SECRETSTORAGE

1. The depositor executes NIZK's KeyGen protocol to obtain a CRS: $\sigma \leftarrow \mathsf{KeyGen}(1^k)$.
2. The depositor computes hash $\mathsf{requestHash} \leftarrow H(F, \sigma)$, and publishes $\mathsf{requestHash}$ on the blockchain. Let id be the storage identifier of the published request.
3. The depositor stores the tuple $(id, (F, \sigma))$ offchain.
4. The depositor and the current members of the miner committee engage in the DPSS **Setup Phase** protocol for the user's secret M.
5. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain, (F, σ) from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of (F, σ):

   $$\mathsf{requestHash} \stackrel{?}{=} H(F, \sigma)$$

   If this is not the case, the committee member stops processing this request.
6. Otherwise, C_i stores $(id, \mathsf{dpss\text{-}data}_i)$ internally, where $\mathsf{dpss\text{-}data}_i$ is the DPSS data obtained from the DPSS **Setup Phase** procedure.

property of our DPSS scheme to switch from honestly executing the DPSS protocol to using a DPSS simulator. At this point, we can show that either the adversary was able to provide a valid secret release request for the challenge secret release function, in which case we are able to extract a witness from the provided NIZK proof (relying on the NIZK's proof-of-knowledge property), or the adversary did not provide a valid secret release request and in this case we are able to "forget" the secret altogether, since it is never used.

Protocol 7: SecretsHandoff

1. For each secret storage identifier id, the miners of the old and the new committee engage in the DPSS **Handoff Phase** for the corresponding secret. Let $\mathsf{dpss\text{-}data}_i^{id}$ denote the resulting DPSS data corresponding to the storage identifier id of party C_i of the new committee after the handoff phase.
2. For each secret storage identifier id, each miner of the new committee stores $(id, \mathsf{dpss\text{-}data}_i^{id})$ internally.

5 Application examples

In this section, we first present some motivational application examples and briefly explain the key ideas behind implementing each of them using our construction. Then, we discuss a specific application - voting - in more detail.

Time-lock encryption. Time-lock encryption, related to timed-release encryption introduced by Rivest et al. [RSW96], allows one to encrypt a message such that it can only be decrypted after a certain deadline has passed. As mentioned by Liu et al. [LJKW18], time-lock encryption must satisfy the following properties:

- The encrypter is not required to be available for decryption.
- The are no trusted third parties.
- Starting to decrypt before the deadline does not provide any advantage, i.e., a party that started to decrypt before the deadline will succeed at some time t after the deadline, same time as the party that started to decrypt directly after the deadline has passed.

Time-lock encryption can be easily implemented using our PublicWitness protocol (see Appendix F) as follows: the encrypter executes SecretStorage request with a secret release condition F specifying the time t when the message can be released. Once the time has passed, a user that wishes to see the message submits a SecretRelease request with the witness "The deadline has passed". Miners then check whether the time is indeed past t and if so, release their fragments of the secret as specified by SecretRelease. Note that with only a slight modification to our protocol it is also possible to allow for automatic decryption - upon receiving a secret storage request with an "automatic" tag miners would place the identifier in a list and periodically check whether the secret release condition holds for any request in this list. Furthermore, note that our construction evades the issue that some time-lock encryption schemes [LJKW18] have: even if the adversary becomes computationally more powerful, it does not allow him to receive the secret message earlier. Additionally, we avoid resource waste, since we do not impose a high overhead on the requester.

Dead man's switch. A dead man's switch is designed to be activated or deactivated when the human operator becomes incapacitated. Software versions of the dead man's switch typically trigger a process such as making public or deleting some data. The triggering event for centralized

Protocol 8: SECRETRELEASE

1. For the desired secret storage request with the identifier id, the requester retrieves $\mathsf{requestHash}$ from the blockchain, (F, σ) from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of (F, σ):
$$\mathsf{requestHash} \overset{?}{=} H(F, \sigma)$$
If this is not the case, the requester aborts.

2. The requester computes a NIZK proof of knowledge of the witness for F and his party identifier p_{id}:
$$\pi \leftarrow P(\sigma, p_{id}, w),$$

3. The requester computes hash of the storage identifier, his party identifier and the proof to obtain $\mathsf{requestHash}^* \leftarrow H(id, p_{id}, \pi)$ and publishes $\mathsf{requestHash}^*$ on blockchain. Let id^* be the identifier of the published request.

4. The requester stores the tuple $(id^*, (id, p_{id}, \pi))$ offchain.

5. Each committee member retrieves $\mathsf{requestHash}^*$ from the blockchain request with the identifier id^*, (id, p_{id}, π) from the offchain storage, and verifies that $\mathsf{requestHash}^*$ is the hash of (id, p_{id}, π):
$$\mathsf{requestHash}^* \overset{?}{=} H(id, p_{id}, \pi)$$
If not, the committee member aborts.

6. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain request with the identifier id, (F, σ) from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of (F, σ):
$$\mathsf{requestHash} \overset{?}{=} H(F, \sigma)$$
If not, the committee member aborts.

7. Each committee member C_i retrieves its share of the secret $\mathsf{pss\text{-}data}_i$ from its internal storage.

8. Each committee member C_i checks whether π is a valid proof using NIZK's verification algorithm V:
$$V(\sigma, p_{id}, \pi) \overset{?}{=} true$$
If so, C_i and the user with the party identifier p_{id} engage in the DPSS **Reconstruction Phase** protocol using $\mathsf{pss\text{-}data}_i$.

Protocol 9: eWEB

1. A user can initiate secret storage by executing SecretStorage-HiddenWitness (6).
2. A user can initiate secret release by executing SecretRelease-HiddenWitness (8) and then obtain the secret using SecretReconstruction procedure (30).
3. In each round, miners execute SecretsHandoff (7) procedure.

software versions can be failing to log in for three days, a GPS signal of a mobile phone that is not moving for a period of time, or not responding to an automated email. A dead man's switch can be seen as insurance for journalists and whistleblowers. It can be implemented using our PublicWitness protocol as follows: the person who wishes to setup the switch generates a SecretStorage request with the desired secret release condition F. Such release condition can be failing to post a signed message on the blockchain for several days or anything publicly verifiable. As in time-lock encryption, it is possible to either use the standard protocol where a person (e.g., a relative or a friend) proves to the miners that the release condition has been satisfied or specify an "automatic" request where the miners periodically check the secret release condition.

Fair MPC. Multi-party computation (MPC) is considered fair if it ensures that either all parties receive the output of the protocol, or none. In the standard model fair MPC was proven to be impossible to achieve for general functions when a majority of the parties are dishonest [Cle86]. Using the witness encryption-based construction of Choudhuri et al. [CGJ^{+}17] that utilizes a public bulletin board model, fair MPC can be implemented by replacing the extractable witness encryption with our eWEB protocol. Conveniently, one of the easiest ways to realize Choudhuri et al.'s public bulletin board in practice is to use a blockchain-based ledger. Thus, by replacing the witness encryption scheme with our blockchain-based protocol, we are not adding any extra assumptions to Choudhuri et al.'s construction.

One-time programs. A one-time program is a program that runs only once and then "self-destructs". This notion was introduced by Goldwasser et al. [GKR08]. In the same work they presented a proof of concept construction that relies on tamper proof hardware. A number of works on one-time programs followed [GIS^{+}10, BHR12, AIKW15, DDKZ13], all of them relied on tamper proof hardware. Goyal and Goyal [GG17] presented a first construction for one-time programs that did not rely on tamper proof hardware (but did rely on extractable witness encryption). As with fair MPC and Choudhuri et al.'s construction, by replacing the witness encryption scheme with our eWEB protocol in Goyal and Goyal's work, we are not adding any extra assumptions since their work already relies on blockchains. Note, however, that their construction necessarily requires POS blockchains.

Non-repudiation/Proof of receipt. Repudiation is defined as a denial by one of the entities involved in a communication of having participated in all or part of the communication. With our eWEB protocol, it is easy to provide a proof that a person received certain data. In this case, the user providing the data stores it using the SecretStorage protocol. To satisfy the release condition F, a user with public identifier *pid* publishes a signed message "User *pid* wishes to receive the message". The miners then securely release a secret to the user *pid* as specified by SecretRelease. The publicly verifiable signature on the message "User *pid* wishes to receive the message" then serves as a proof that party *pid* indeed received the data.

5.1 Voting Protocol

As a more detailed example, we consider the "yes-no" voting scenario. Specifically, the space of the secrets is $\{-1, 1\}$, where -1 stands for a "no" vote, and 1 stands for a "yes" vote. Clients vote independently at different times, miners store their votes while the voting takes place and release the sum of all votes at a predefined time. The vote of any particular client must be kept private and no client should be able to manipulate the outcome more than with his "yes-no" vote. The last requirement requires the committee to be able to verify the correctness of the secrets shared by the clients, i.e., that $s \in \{-1, 1\}$. Note that proving $s \in \{-1, 1\}$ is equivalent to proving $s^2 = 1$. While this can be done using the commitment to s, the client would additionally need to prove that the committed value is the same as the one that client shared to the committee. To avoid this expensive check, the members of the committee prepare the commitments for clients by themselves. We show that all parties can prepare the Pedersen commitments [Ped92] for clients with constant amortized cost (see Appendix G.1 for details). These commitments are of the form $g^s h^z$, where z is a random value and g, h are two random group generators. Note that the client knows not only g, but also g, h and z.

To prove that $s^2 = 1$, the client computes $w = g^{2sz} h^{z^2}$ and publishes w to all parties. Note that it can be done since the client knows s and z. Let β satisfy that $h = g^\beta$. To verify whether $s^2 = 1$, all parties check the following equation:

$$e(g^s h^z, g^s h^z) = e(g, g) \cdot e(h, w).$$

Correctness. To show correctness, note that

$$\begin{aligned} \text{LHS} &= e(g^{s+\beta z}, g^{s+\beta z}) = e(g, g)^{s^2+2\beta sz+\beta^2 z^2} \\ \text{RHS} &= e(g, g) \cdot e(g^\beta, g^{2sz+\beta z^2}) = e(g, g)^{1+2\beta sz+\beta^2 z^2}. \end{aligned}$$

Therefore, if the equation holds, then we have $s^2 = 1$ and thus the vote submitted by the client is valid.

To compute the result of the voting procedure the committee can compute the sharing of the result relying on the linear homomorphism of Shamir's secret sharing scheme and KZG commitment scheme, and then follow the usual SecretRelease procedure.

6 Related work

Since we believe that our DPSS construction may be of independent interest, we first discuss previous proactive secret sharing protocols. Then, we elaborate on known approaches for extractable witness encryption.

6.1 Prior Work on DPSS

Since the introduction of proactive secret sharing by Herzberg et al. [HJKY95], a number of PSS schemes have been developed. These schemes vary in terms of security guarantees, network assumptions (synchronous or asynchronous), communication complexity and whether or not they can handle dynamic changes in the committee membership. In Figure 1 we provide an overview of PSS schemes and compare the relevant parameters.

Below, we compare our DPSS construction in detail with the two constructions (CHURP [MZW$^+$19], Baron et al. [BDLO15]) that are most closely related to our protocol, as they are also the most efficient DPSS schemes to date.

In the best case, CHURP [MZW$^+$19] achieves communication complexity $O(n^2)$ plus $O(n) \cdot \mathcal{B}$ to refresh each secret in the hand-off phase, where n is the number of parties and $\mathcal{B}$ denotes the cost of broadcasting a bit. In the worst case (where some corrupted party deviates from the protocol), it requires $O(n^2) \cdot \mathcal{B}$ communication per secret.

In the single-secret setting, our protocol achieves the same asymptotic communication complexity as CHURP. However, our protocol achieves *amortized* communication complexity $O(n)$ plus $O(1) \cdot \mathcal{B}$ per secret in the best case, and $O(n^2)$ plus $O(n) \cdot \mathcal{B}$ per secret in the worst case. Batched setting is very relevant to our eWEB application since there might be thousands of secrets stored at any given time.

In terms of techniques, while CHURP uses asymmetric bivariate polynomials to refresh a secret in the hand-off phase, we use a modified version of the technique borrowed from Damgård et al. [DN07] to prepare a batch of random secret sharings which will be used in the hand-off phase. We note that generating random secret sharings is much more efficient than generating bivariate polynomials. Specifically, each bivariate polynomial requires $O(n)$ communication per party; i.e., $O(n^2)$ in total. On the other hand, we can prepare $O(n)$ random sharings with the same communication cost as preparing one bivariate polynomial. To benefit from it, we use a entirely different way to refresh each secret.

The work of Baron et al. [BDLO15] focuses on a slightly different setting from ours. Specifically, they are interested in constructing DPSS protocols with unconditional security and a $(1/2 - \epsilon)$ corruption threshold, where ϵ is a constant. The construction achieves amortized communication complexity $O(1)$ per secret. However, in the single-secret setting, it requires $O(n^3)$ communication to refresh a secret.

Because Baron et al. consider a $(1/2 - \epsilon)$ corruption threshold, they are able to utilize the technique of packed secret sharing [FY92], which allows to store a batch of $O(n)$ secrets in one sharing by encoding multiple secrets as distinct points of a single polynomial. Therefore, refreshing each sharing effectively refreshes a batch of $O(n)$ secrets at the same time. However, with this construction, secrets in the same batch can only be refreshed or reconstructed together. On the other hand, our construction only generates random sharings in a batch. After that, each secret is refreshed individually.

6.2 Extractable Witness Encryption

The notion of witness encryption was introduced by Garg et al. [GGSW13]. The notion of extractable security for witness encryption schemes was first proposed by Goldwasser et al. [GKP$^+$13]. In their work a candidate construction was introduced that requires very strong assumptions over multilinear maps. As mentioned by Liu et al. [LJKW18], existing witness encryption schemes have no efficient extraction methods. The work of Garg et al. [GGHW14] suggests that it even might be impossible to achieve extractable witness encryption with arbitrary auxiliary inputs.

Nevertheless, as mentioned in Sections 1 and 5 the notion of extractable witness encryption has been extensively used in various cryptographic protocols [CGJ$^+$17, GG17, BH15, LJKW18], especially in conjunction with blockchains.

7 Conclusion

We have introduced a new cryptographic primitive: an extractable witness encryption on blockchain (eWEB), which allows to store and release secrets on blockchain. We provided a proof of concept eWEB protocol. A key building block of this protocol is a highly efficient batched dynamic secret sharing scheme that may be of independent interest.

References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013.

[AIKW15] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate, or how to compress garbled circuit keys. *SIAM Journal on Computing*, 44(2):433–466, 2015.

[BDLO15] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security*, pages 23–41, Cham, 2015. Springer International Publishing.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM)*, 59(2):1–48, 2012.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 416–432. Springer, 2003.

[BH15] Mihir Bellare and Viet Tung Hoang. Adaptive witness encryption and asymmetric password-based cryptography. In *IACR International Workshop on Public Key Cryptography*, pages 308–331. Springer, 2015.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 134–153. Springer, 2012.

[BSFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[CGJ$^+$17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 719–728, 2017.

[CHM$^+$19] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Report 2019/1047, 2019. https://eprint.iacr.org/2019/1047.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

[DDKZ13] Konrad Durnoga, Stefan Dziembowski, Tomasz Kazana, and Michal Zajac. One-time programs with limited memory. In *International Conference on Information Security and Cryptology*, pages 377–394. Springer, 2013.

[DJ97] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Citeseer, 1997.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[FY92] Matthew Franklin and Moti Yung. Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710, 1992.

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In *Theory of Cryptography Conference*, pages 529–561. Springer, 2017.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *Annual Cryptology Conference*, pages 518–535. Springer, 2014.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476. ACM, 2013.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *Theory of Cryptography Conference*, pages 308–326. Springer, 2010.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Annual Cryptology Conference*, pages 536–553. Springer, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. One-time programs. In *Annual International Cryptology Conference*, pages 39–56. Springer, 2008.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 444–459. Springer, 2006.

[HJKY95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Annual International Cryptology Conference*, pages 339–352. Springer, 1995.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[LJKW18] Jia Liu, Tibor Jager, Saqib A Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, 2018.

[MZW$^+$19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: Dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2369–2386, 2019.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[SLL08] David A Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 458–458, 2008.

[WWW02] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 94–105. IEEE, 2002.

[ZSVR05] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. Apss: Proactive secret sharing in asynchronous systems. In *ACM transactions on information and system security (TISSEC)*, pages 259–286, 2005.

A Preliminaries

In this section, we introduce cryptographic building blocks used in our system.

A.1 Encryption schemes

In our protocol, we rely on public key encryption schemes. We use a standard definition of a public key encryption scheme consisting of a key generation algorithm *Gen*, an encryption algorithm *Enc*, and a decryption algorithm *Dec* that satisfy the correctness property. For security purposes, in our scheme we rely on **multi-message IND-CCA security**, defined as follows:

Definition A.1. Multi-message IND-CCA security

An encryption scheme (Gen, Enc, Dec) is said to be IND-CCA secure if any PPT adversary $\mathcal{A}$ is successful in the following game with the probability at most $\frac{1}{2} + negl(n)$, where $negl(\cdot)$ is a negligible function:

- The challenger generates keys $(pk, sk) \leftarrow Gen(1^n)$.
- $\mathcal{A}$ receives pk as input.
- The challenger chooses $b \leftarrow \{0, 1\}$ at random.
- $\mathcal{A}$ gets a black-box access to $Dec_{sk}(\cdot)$.
- In each round i until $\mathcal{A}$ wishes to end the game the following happens:
  - $\mathcal{A}$ chooses m_i.
  - The challenger gives $\mathcal{A}$ the challenge ciphertext $c_i = Enc_{pk}(m_i)$ if $b = 0$, and $c_i = Enc_{pk}(0)$ otherwise.
  - $\mathcal{A}$'s access to $Dec_{sk}(\cdot)$ is now restricted - $\mathcal{A}$ is not allowed to ask for the decryption of c_i.
- $\mathcal{A}$ outputs $b^* \in \{0, 1\}$ and wins if $b = b^*$.

Note that the multi-message IND-CCA security is equivalent to the (single-message) IND-CCA security.

A.2 Non-interactive zero-knowledge proofs

Non-interactive zero-knowledge proofs (NIZKs) allow one party (the prover) prove validity of some statement to another party (the verifier), in a way that nothing except for the validity of the statement is revealed and no interaction between the prover and the verifier is needed. More formally, as defined in [Gro06]:

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness.

A proof system for relation R consists of a key generation algorithm $KeyGen$, a prover P and a verifier V. The key generation algorithm produces a CRS σ. The prover takes as input (σ, x, w) and produces a proof π. The verifier takes as input (σ, x, π) and outputs 1 if the proof is acceptable and 0 otherwise.

Definition A.2. Proof system

We say $(KeyGen, P, V)$ is a proof system for R if it satisfies the following properties:

Perfect completeness. For all adversaries $\mathcal{A}$ holds:

$$\begin{gathered} Pr[\sigma \leftarrow KeyGen(1^k); (x, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow P(\sigma, x, w) : \\ V(\sigma, x, \pi) = 1 \text{ if } (x, w) \in R] = 1 \end{gathered}$$

Perfect soundness. For all adversaries $\mathcal{A}$ holds:

$$\begin{gathered} Pr[\sigma \leftarrow KeyGen(1^k); (x, \pi) \leftarrow \mathcal{A}(\sigma) : \\ V(\sigma, x, \pi) = 0 \text{ if } x \notin L] = 1 \end{gathered}$$

We call $(KeyGen, P, V)$ a **proof of knowledge** for R if there exists a polynomial time extractor $E = (E_1, E_2)$ with the following properties:

For all adversaries $\mathcal{A}$ holds:

$$\begin{gathered} Pr[\sigma \leftarrow KeyGen(1^k) : \mathcal{A}(\sigma) = 1] = \\ Pr[(\sigma, \epsilon) \leftarrow E_1(1^k) : \mathcal{A}(\sigma) = 1], \text{ and} \\ Pr[(\sigma, \epsilon) \leftarrow E_1(1^k); (x, \pi) \leftarrow \mathcal{A}(\sigma); w \leftarrow E_2(\sigma, \epsilon, x, \pi) : \\ V(\sigma, x, \pi) = 0 \text{ or } (x, w) \in R] = 1 \end{gathered}$$

Definition A.3. NIZK

We call a non-interactive proof system $(KeyGen, P, V)$ a **non-interactive zero-knowledge proof (NIZK)** for R if there exists a polynomial time simulator $S = (S_1, S_2)$, which satisfies the following property:

(Unbounded) computational zero-knowledge. For all PPT adversaries $\mathcal{A}$ holds

$$Pr[\sigma \leftarrow KeyGen(1^k) : \mathcal{A}^{P(\sigma,\cdot,\cdot)}(\sigma) = 1] \approx Pr[(\sigma, \tau) \leftarrow S_1(1^k) : \mathcal{A}^{S(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1],$$

where $f(k) \approx g(k)$ means that there exists a negligible function $negl(k)$ s.t. $|f(k) - g(k)| < negl(k)$

Finally, in our protocol we require that if after seeing some number of simulated proofs the adversary is able to produce a new valid proof, we are able to extract a witness from this proof. This property is called simulation sound extractability [Gro06]:

(Unbounded) simulation sound extractability. We say a NIZK proof of knowledge $(KeyGen, P, V, S_1, S_2, E_1, E_2)$ is simulation sound extractable, if for all PPT adversaries $\mathcal{A}$ holds

$$\begin{gathered} Pr[(\sigma, \tau, \epsilon) \leftarrow SE_1(1^k); (x, \pi) \leftarrow \mathcal{A}^{S_2(\sigma,\tau,\cdot)}(\sigma, \epsilon); w \leftarrow E_2(\sigma, \epsilon, x, \pi) : (x, \pi) \notin Q \text{ and } (x, w) \notin \\ R \text{ and } V(\sigma, x, \pi) = 1] \approx 0, \end{gathered}$$

where SE_1 is an algorithm that outputs (σ, τ, ϵ) such that it is identical to S_1 when restricted to the first two parts (σ, τ), and Q is a list of simulation queries and responses (i.e., simulated proofs).

A.3 Hash functions

To reduce communication cost of our protocol, we use hash functions. In the following, we formally define a hash function family and the collision-resistance property that we rely on in our construction.

Definition A.4. Hash function family

A family of functions $H = \{h_i : D_i \to R_i\}_{i \in I}$ is a hash function family, if the following holds:

Easy to sample. There exists a PPT algorithm Gen outputting i s.t. h_i is a random member of H.

Easy to evaluate. There exists a PPT algorithm $Eval$ s.t. for all $i \in I$ and $x \in D_i$, $Eval(i, x) = h_i(x)$.

Compression. For all $i \in I$ holds $|R_i| < |D_i|$.

We say a hash function family is a collision resistant hash function (CRHF) if the following condition holds:

Collision-resistance. Foll all PPT $\mathcal{A}$, security parameter k and a negligible function $negl(\cdot)$ holds:

$$Pr[i \leftarrow Gen(k); (x, x') \leftarrow \mathcal{A}(i) : x \neq x' \text{ and } h_i(x) = h_i(x')] \leq negl(k)$$

A.4 Shamir Secret Sharing

As a part of our DPSS construction, we use the standard Shamir secret sharing scheme [Sha79].

For a finite field $\mathbb{F}$, a *degree-d* Shamir sharing of $w \in \mathbb{F}$ is a vector $(w_1, \dots, w_n)$ which satisfies that, there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(0) = w$ and $f(i) = w_i$ for $i \in \{1, \dots, n\}$. Each party P_i holds a share w_i and the whole sharing is denoted as $[w]_d$.

A.5 Polynomial Commitment Schemes

In essence, a degree-d Shamir sharing is a degree-d polynomial. In our DPSS construction we will use polynomial commitment schemes to transform a plain Shamir sharing to a verifiable secret sharing. The following definition of the polynomial commitment scheme comes from [KZG10]. For simplicity, we only focus on perfectly hiding commitment schemes.

Definition. A polynomial commitment scheme contains the following 6 algorithms.

- $\mathsf{Setup}(1^\kappa, t)$: It generates a pair of public-private keys $(\mathrm{PK}, \mathrm{SK})$ to commit a polynomial of degree at most t in $\mathbb{F}$. SK is not required in the rest of the scheme.
- $\mathsf{Commit}(\mathrm{PK}, f(\cdot))$: It outputs a commitment $\mathcal{C}om$ to the polynomial $f(\cdot)$ and some associated decommitment information d.
- $\mathsf{Open}(\mathrm{PK}, \mathcal{C}om, f(\cdot), d)$: It outputs the polynomial $f(\cdot)$ used to create the commitment $\mathcal{C}om$, with decommitment information d.
- $\mathsf{VerifyPoly}(\mathrm{PK}, \mathcal{C}om, f(\cdot), d)$: It verifies the correctness of the opening. If the opening is correct, it outputs 1. Otherwise, it outputs 0.
- $\mathsf{CreateWitness}(\mathrm{PK}, f(\cdot), i, d)$: It outputs $(i, f(i), w_i)$ where w_i is a witness to check the correctness of $f(i)$.
- $\mathsf{VerifyEval}(\mathrm{PK}, \mathcal{C}om, i, f(i), w_i)$: It verifies the correctness of the evaluation $f(i)$. If the evaluation is correct, it outputs 1. Otherwise, it outputs 0.

Definition A.5 ([KZG10]). We say $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyPoly}, \mathsf{CreateWitness}, \mathsf{VerifyEval})$ is a secure polynomial commitment scheme if it satisfies the following properties.

Correctness. For all $f(\cdot) \in \mathbb{F}[X]$, let $\mathrm{PK} \leftarrow \mathsf{Setup}(1^\kappa, t)$ and $\mathcal{C}om \leftarrow \mathsf{Commit}(\mathrm{PK}, f(\cdot))$.

- The output of $\mathsf{Open}(\mathrm{PK}, \mathcal{C}om, f(\cdot), d)$ is successfully verified by $\mathsf{VerifyPoly}(\mathrm{PK}, \mathcal{C}om, f(\cdot), d)$.
- For all $(i, f(i), w_i) \leftarrow \mathsf{CreateWitness}(\mathrm{PK}, f(\cdot), i, d)$, it is successfully verified by $\mathsf{VerifyEval}(\mathrm{PK}, \mathcal{C}om, i, f(i), w_i)$.

Polynomial Binding. There exists a negligible function $\epsilon(\cdot)$ such that for all adversaries $\mathcal{A}$:

$$\begin{aligned}\Pr[\quad & \mathrm{PK} \leftarrow \mathsf{Setup}(1^\kappa, t), (\mathcal{C}om, f(\cdot), f'(\cdot), d, d') \leftarrow \mathcal{A}(\mathrm{PK}) : \\ & \mathsf{VerifyPoly}(\mathrm{PK}, \mathcal{C}om, f(\cdot), d) = 1 \text{ and} \\ & \mathsf{VerifyPoly}(\mathrm{PK}, \mathcal{C}om, f'(\cdot), d') = 1 \text{ and } f \neq f'] \leq \epsilon(\kappa).\end{aligned}$$

Evaluation Binding. There exists a negligible function $\epsilon(\cdot)$ such that for all adversaries $\mathcal{A}$:

$$\begin{aligned}\Pr[\quad & \text{PK} \leftarrow \mathsf{Setup}(1^\kappa, t), (\mathcal{C}om, \langle i, f(i), w_i\rangle, \langle i, f'(i), w'_i\rangle) \leftarrow \mathcal{A}(\text{PK}) : \\ & \mathsf{VerifyEval}(\text{PK}, \mathcal{C}om, i, f(i), w_i) = 1 \text{ and} \\ & \mathsf{VerifyEval}(\text{PK}, \mathcal{C}om, i, f'(i), w'_i) = 1 \text{ and } f(i) \neq f'(i)] \leq \epsilon(\kappa).\end{aligned}$$

Hiding. Given $(\text{PK}, \mathcal{C}om)$ and $\{(i_j, f(i_j), w_{i_j})\}_{j\in[t]}$ for $f(\cdot) \in \mathbb{F}[X]$ such that $\mathsf{VerifyEval}(\text{PK}, \mathcal{C}om, i_j, f(i_j), w_{i_j}) = 1$ for all $j \in [t]$, no computationally unbounded adversary $\mathcal{A}$ has any information about $f(k)$ where $k \notin \{i_1, \ldots, i_t\}$.

KZG Commitment Scheme. The following commitment scheme is from [KZG10].

- $\mathsf{Setup}(1^\kappa, t)$: It generates two groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order p (with $p \geq 2^\kappa$) such that there exists a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $t-\text{SDH}$ assumption holds. Then it randomly samples two generators $g, h \in \mathbb{G}$ and $\alpha \in [p-1]$. The secret key SK is α and the public key PK is $(\mathbb{G}, \mathbb{G}_T, e, g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^t}, h, h^\alpha, h^{\alpha^2}, \ldots, h^{\alpha^t})$. The field $\mathbb{F}$ is set to be $\mathbb{Z}_p$.
- $\mathsf{Commit}(\text{PK}, f(\cdot))$: It chooses a random polynomial $r(\cdot) \in \mathbb{Z}_p[X]$ of degree t. Then compute the commitment $\mathcal{C}om = g^{f(\alpha)}h^{r(\alpha)}$ using PK. Set $d = r(\cdot)$.
- $\mathsf{Open}(\text{PK}, \mathcal{C}om, f(\cdot), r(\cdot))$: It outputs $f(\cdot), r(\cdot)$.
- $\mathsf{VerifyPoly}(\text{PK}, \mathcal{C}om, f(\cdot), r(\cdot))$: If the degree of either $f(\cdot)$ or $r(\cdot)$ is larger than t, it outputs 0. Otherwise, it computes $g^{f(\alpha)}h^{r(\alpha)}$ using PK and compares with $\mathcal{C}om$. If the result matches $\mathcal{C}om$, it outputs 1. Otherwise, it outputs 0.
- $\mathsf{CreateWitness}(\text{PK}, f(\cdot), i, r(\cdot))$: It first computes $f'(x) = \frac{f(x)-f(i)}{x-i}$ and $r'(x) = \frac{r(x)-r(i)}{x-i}$. Then compute $w_i = g^{f'(\alpha)}h^{r'(\alpha)}$ using PK. Finally, output $(i, f(i), r(i), w_i)$.
- $\mathsf{VerifyEval}(\text{PK}, \mathcal{C}om, i, f(i), r(i), w_i)$: It checks whether $e(\mathcal{C}om, g) = e(w_i, g^\alpha/g^i)e(g^{f(i)}h^{r(i)}, g)$. If true, it outputs 1. Otherwise, it outputs 0.

Definition A.6 (t-Strong Diffie-Hellman (t-SDH) Assumption [KZG10]). Let α be a random element in $\mathbb{Z}_p^*$. Given as input a $(t+1)$-tuple $\langle g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^t}\rangle \in \mathbb{G}^{t+1}$, for every adversary $\mathcal{A}_{t\text{-SDH}}$, the probability

$$\Pr[\mathcal{A}_{t\text{-SDH}}(g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^t}) = \langle c, g^{\frac{1}{\alpha+c}}\rangle]$$

is negligible for any value $c \in \mathbb{Z}_p \backslash \{-\alpha\}$.

Lemma A.1 ([KZG10]). The above scheme is a secure polynomial commitment scheme provided the $t-\text{SDH}$ assumption holds in $(e, \mathbb{G}, \mathbb{G}_T)$.

Linear Homomorphism of the KZG Commitment Scheme. In the following, we use $\mathcal{C}om_{\text{KZG}}(f; r)$ to denote the commitment of $f(\cdot)$ that uses a polynomial $r(\cdot)$ as randomness. We use $w_i(f; r)$ to denote the witness associated with the evaluation point i. Concretely,

$$\mathcal{C}om_{\text{KZG}}(f; r) = g^{f(\alpha)}h^{r(\alpha)},\ w_i(f; r) = g^{\frac{f(\alpha)-f(i)}{\alpha-i}}h^{\frac{r(\alpha)-r(i)}{\alpha-i}}.$$

It is easy to verify that, for any constants c_1, c_2 and polynomials f_1, f_2, r_1, r_2,

$$\begin{gathered}\mathcal{C}om_{\mathrm{KZG}}(c_1 f_1 + c_2 f_2; c_1 r_1 + c_2 r_2) = \\ (\mathcal{C}om_{\mathrm{KZG}}(f_1; r_1))^{c_1} \cdot (\mathcal{C}om_{\mathrm{KZG}}(f_2; r_2))^{c_2}, \\ w_i(c_1 f_1 + c_2 f_2; c_1 r_1 + c_2 r_2) = (w_i(f_1; r_1))^{c_1} \cdot (w_i(f_2; r_2))^{c_2}.\end{gathered}$$

A.6 Preparing Random Sharings

The following protocol comes from [DN07], which allows a set of parties to prepare a batch of random sharings. The protocol will utilize a predetermined and fixed Vandermonde matrix of size $n \times (n-t)$, which is denoted by $\boldsymbol{M}^{\mathrm{T}}$ (therefore $\boldsymbol{M}$ is a $(n-t) \times n$ matrix). An important property of a Vandermonde matrix is that any $(n-t) \times (n-t)$ submatrix of $\boldsymbol{M}^{\mathrm{T}}$ is *invertible*. Therefore, given the sharings generated by corrupted parties, there is a one-to-one map from the first $(n-t)$ random sharings generated by honest parties to the resulting $(n-t)$ sharings. The description of Rand appears in Protocol 10. The communication complexity of Rand is $O(n^2)$ field elements.

Protocol 10: Rand

1. Each party C_i randomly samples a sharing $[s^{(i)}]_t$ and distributes the shares to other parties.
2. All parties locally compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \dots, [r^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([s^{(1)}]_t, [s^{(2)}]_t, \dots, [s^{(n)}]_t)^{\mathrm{T}}$$

and output $[r^{(1)}]_t, [r^{(2)}]_t, \dots, [r^{(n-t)}]_t$.

Lemma A.2 ([DN07]). For a semi-honest adversary $\mathcal{A}$, $r^{(1)}, \dots, r^{(n-t)}$ are uniformly random given the views of $\mathcal{A}$.

A.7 Generating Challenge

We introduce a simple protocol Challenge, which comes from [BSFO12], to let all parties generate an element in $\mathbb{F}$ with high min-entropy. The cost of Challenge is $O(\kappa)$ on-chain communication.

Protocol 11: Challenge

1. Each party P_i chooses a random string $\mathtt{str}_i \in \{0,1\}^{\frac{\log|\mathbb{F}|}{n}}$ and publish $\mathtt{str}_i$.
2. All parties convert $(\mathtt{str}_1, \dots, \mathtt{str}_n)$ into an element λ in $\mathbb{F}$.

Lemma A.3 ([BSFO12]). For any fixed set of t corrupt parties and for any given subset $\mathcal{S} \in \mathbb{F}$, the probability that a challenge generated by CHALLENGE lies in $\mathcal{S}$ is at most

$$\frac{|\mathcal{S}|}{2^{(n-t)\frac{\log|\mathbb{F}|}{n}}} \leq \frac{|\mathcal{S}|}{2^{\kappa/2}}.$$

B Limitation of the KZG Commitment Scheme

Intuitively, the use of a polynomial commitment scheme should prevent the case that a corrupted party provides a wrong share with a valid witness from happening. This is because when the dealer opens the whole sharing, there will be two opens of the commitment for different values at the same evaluation point, which breaks the evaluation binding property. *However, if the dealer is corrupted, it may not even know how to open the commitment properly.* Usually, this problem can be tackled by requiring the polynomial commitment scheme to be extractable, i.e., there exists an extractor which can extract the polynomial the dealer committed.

However, a major limitation of the KZG commitment scheme is that the commitment is not extractable. It turns out that directly adding extractable feature to the KZG commitment scheme is expensive (or requires additional assumptions) [CHM+19]. To avoid such cost, our idea is to additionally verify the computation of the commitments during the verification of couple sharings.

We note that a nice feature of the KZG commitment is that if a commitment is correctly computed, from $t+1$ shares with valid witnesses, one can compute the commitment of this sharing (i.e., the witnesses include all randomness which is necessary to compute the commitment). Therefore, when checking the correctness of couple sharings, all parties also use the shares with valid witnesses to check whether the commitment is correctly computed. Note that this guarantees that *from the shares and witnesses held by honest parties, one can re-compute the whole sharings and valid witnesses for all shares.* It achieves the same effect as the extractable feature. If a corrupted party provides an incorrect share with a valid witness at a later point, then it effectively breaks the evaluation binding property since we have already learned the correct share and the witness of this party from the shares and witnesses held by honest parties.

C Our DPSS Construction

In the following, we describe provide our complete DPSS protocol. We start by explaining the handoff phase (§C.1), then describe the setup phase (§C.2) and the reconstruction phase (§C.3), and give the full protocol in (§C). We provide the proof of security in Appendix D.

C.1 Hand-off Phase

In the following, we will use $\mathcal{C} = \{P_1, \ldots, P_n\}$ to represent the old committee and $\mathcal{C}' = \{P'_1, \ldots, P'_n\}$ to represent the new committee.

The hand-off phase includes two steps, preparation phase and refresh phase.

- In the preparation phase, the new committee will prepare two degree-t sharings of the same random value $r(=\tilde{r})$, denoted by $[r]_t$ and $[\tilde{r}]_t$. The new committee will hold the shares of $[\tilde{r}]_t$ while the old committee will hold the shares of $[r]_t$. We refer to these two sharings as a pair of *couple sharings.*

- In the refresh phase, the old committee will reconstruct the sharing $[s]_t + [r]_t$ and publish the result. The new committee will set $[\tilde{s}]_t = (s + r) - [\tilde{r}]_t$. Since $r = \tilde{r}$, we have $s = \tilde{s}$.

Preparing Couple Sharings with Polynomial Commitments The couple sharings are prepared in a batch way. Recall that each Shamir sharing itself is a polynomial. In the following, we use a sharing $[s]_t$ to also denote its corresponding polynomial. We use $\mathcal{C}om_{\mathrm{KZG}}([s]_t; [z]_t)$ to denote the KZG commitment of $[s]_t$ using the random polynomial $[z]_t$. Let $w_i([s]_t; [z]_t)$ denote the witness output by $\mathsf{CreateWitness}(\mathrm{PK}, [s]_t, i, [z]_t)$.

Distribution. Each party P'_i in the new committee first prepares two pairs of random couple sharings $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$, $([v^{(i)}]_t, [\tilde{v}^{(i)}]_t)$. The second pair of random couple sharings are used as random polynomials when committing the first pair of random couple sharings. Concretely, each P'_i computes $\mathcal{C}om_{\mathrm{KZG}}([u^{(i)}]_t; [v^{(i)}]_t)$ and $\mathcal{C}om_{\mathrm{KZG}}([\tilde{u}^{(i)}]_t; [\tilde{v}^{(i)}]_t)$.

Then for each $P_j \in \mathcal{C}$, P'_i sends P_j the j-th shares of $[u^{(i)}]_t, [v^{(i)}]_t$ and the associated witness $w_j([u^{(i)}]_t; [v^{(i)}]_t)$. For each $P'_j \in \mathcal{C}'$, P'_i sends P'_j the j-th shares of $[\tilde{u}^{(i)}]_t, [\tilde{v}^{(i)}]_t$ and the associated witness $w_j([\tilde{u}^{(i)}]_t; [\tilde{v}^{(i)}]_t)$. The commitments of $[u^{(i)}]_t, [\tilde{u}^{(i)}]_t$ are published.

Finally, all parties verify the validity of their shares and witnesses. The description of DISTRIBUTION appears in Protocol 12. The communication complexity of DISTRIBUTION is $O(n^2)$ elements plus $O(n)$ elements of broadcasting.

Protocol 12: DISTRIBUTION

1. Each party P'_i in the new committee randomly samples two pairs of couple sharings $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ and $([v^{(i)}]_t, [\tilde{v}^{(i)}]_t)$.
2. Each party P'_i in the new committee computes $\mathcal{C}om_{\mathrm{KZG}}([u^{(i)}]_t, [v^{(i)}]_t)$, $\mathcal{C}om_{\mathrm{KZG}}([\tilde{u}^{(i)}]_t, [\tilde{v}^{(i)}]_t)$ and publishes the commitments.
3. Each party P'_i in the new committee distributes $[u^{(i)}]_t, [v^{(i)}]_t$ associated with the witnesses to the old committee $\mathcal{C}$ and distributes $[\tilde{u}^{(i)}]_t, [\tilde{v}^{(i)}]_t$ associated with the witnesses to the new committee $\mathcal{C}'$.
4. All parties verify the validity of their shares and witnesses by invoking $\mathsf{VerifyEval}$ of the KZG commitment scheme.

Accusation and Response. After DISTRIBUTION, all parties run a two-round accusation-response protocol to ensure the success of the distribution of the random couple sharings.

Specifically, if some party $P_j \in \mathcal{C} \bigcup \mathcal{C}'$ finds that the shares and the witness received from P'_i are invalid, P_j publishes $(\texttt{accuse}, P'_i)$. In response, P'_i publishes the shares and the witness which were sent to P_j.

If the shares and the witness published by P'_i are invalid, all parties regard P'_i as a corrupted party. Otherwise, P_j will use the new shares and witness. Note that an honest party never accuses another honest party. Therefore, either P_j is corrupted or P'_i is corrupted. In either case, the shares and the witness are known to the adversary and therefore are free to be published.

Note that after this step, a party who fails to provide a share with a valid witness of some sharing prepared in DISTRIBUTION must be corrupted. The description of ACCUSATION-RESPONSE appears in Protocol 13. Note that in the best case where all parties behave honestly, there is no accusation and ACCUSATION-RESPONSE requires no communication. In the worst case, each party may accuse $O(n)$ parties, resulting in $O(n^2)$ elements of broadcasting.

Protocol 13: ACCUSATION-RESPONSE

1. For each party $P_j \in \mathcal{C} \bigcup \mathcal{C}'$, if the shares and the witness received from $P_i' \in \mathcal{C}'$ are invalid, P_j publishes $(\texttt{accuse}, P_i')$.
2. For each accusation against P_i' made by P_j, P_i' publishes the shares and the witness which were sent to P_j.
3. For each accusation against P_i' made by P_j, all parties check the validity of the shares and the witness published by P_i'.
   - If the shares and the witness are invalid, all parties regard P_i' as a corrupted party.
   - Otherwise, P_j uses the shares and the witness published by P_i'.

Verification. Now, we need to check whether the commitments are correctly computed and whether $u = \tilde{u}$ and $v = \tilde{v}$.

All parties first invoke DISTRIBUTION and ACCUSATION-RESPONSE again. Let $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t), ([\nu^{(i)}]_t, [\tilde{\nu}^{(i)}]_t)$ denote the couple sharings generated by $P_i' \in \mathcal{C}'$. Then parties in the old committee hold shares of $[\mu^{(i)}]_t, [\nu^{(i)}]_t$ and valid witnesses. Parties in the new committee hold shares of $[\tilde{\mu}^{(i)}]_t, [\tilde{\nu}^{(i)}]_t$ and valid witnesses. These sharings are used as random masks to protect the secrecy of the original sharings in the following check.

Let $F(\cdot), \tilde{F}(\cdot), H(\cdot), \tilde{H}(\cdot)$ be 4 polynomials of degree $2n-1$ such that

$$F(X) = \sum_{i=1}^{n} (\mu^{(i)} + u^{(i)}X)X^{2i-2}, \quad \tilde{F}(X) = \sum_{i=1}^{n} (\tilde{\mu}^{(i)} + \tilde{u}^{(i)}X)X^{2i-2},$$
$$H(X) = \sum_{i=1}^{n} (\nu^{(i)} + v^{(i)}X)X^{2i-2}, \quad \tilde{H}(X) = \sum_{i=1}^{n} (\tilde{\nu}^{(i)} + \tilde{v}^{(i)}X)X^{2i-2}.$$

Recall that the KZG commitment scheme is a linear homomorphism. Then for all constant λ, the old committee can compute $[F(\lambda)]_t, [H(\lambda)]_t$ and the commitment $\mathcal{C}om_{\text{KZG}}([F(\lambda)]_t; [H(\lambda)]_t)$ with corresponding witnesses. The new committee can compute $[\tilde{F}(\lambda)]_t, [\tilde{H}(\lambda)]_t$ and the commitment $\mathcal{C}om_{\text{KZG}}([\tilde{F}(\lambda)]_t; [\tilde{H}(\lambda)]_t)$ with corresponding witnesses.

Note that if for at least $2n$ points $\{\lambda_i\}_{i \in [2n]}$,

1. $F(\lambda_i) = \tilde{F}(\lambda_i)$ and $H(\lambda_i) = \tilde{H}(\lambda_i)$,
2. the commitments $\mathcal{C}om_{\text{KZG}}([F(\lambda_i)]_t; [H(\lambda_i)]_t)$ and $\mathcal{C}om_{\text{KZG}}([\tilde{F}(\lambda_i)]_t; [\tilde{H}(\lambda_i)]_t)$ are correctly computed,

then all couple sharings and commitments are correctly generated. Therefore, if some party P'_i distributed incorrect couple sharings or provided invalid commitments, we can detect it with overwhelming probability by testing a random evaluation point.

To this end, all parties in the new committee $\mathcal{C}'$ invoke CHALLENGE to generate a challenge $\lambda \in \mathbb{Z}_p$. Parties in the old committee publish their shares of $[F(\lambda)]_t, [H(\lambda)]_t$ and the associated witnesses. Parties in the new committee publish their shares of $[\tilde{F}(\lambda)]_t, [\tilde{H}(\lambda)]_t$ and the associated witnesses. For each sharing, all parties use the first $t+1$ shares that pass the verification to reconstruct the whole sharing. Then check if

1. $F(\lambda) = \tilde{F}(\lambda)$ and $H(\lambda) = \tilde{H}(\lambda)$,
2. the commitments $\mathcal{C}om_{\text{KZG}}([F(\lambda)]_t; [H(\lambda)]_t)$ and $\mathcal{C}om_{\text{KZG}}([\tilde{F}(\lambda)]_t; [\tilde{H}(\lambda)]_t)$ are correctly computed.

If the check fails, all parties take `fail` as output. The description of VERIFICATION appears in Protocol 14. The communication complexity of VERIFICATION is $O(n^2)$ elements plus $O(n)$ elements of broadcasting when all parties behave honestly, and $O(n^2)$ elements plus $O(n^2)$ elements of broadcasting in the worst case.

Lemma C.1. If at least one party P'_i did not generate valid couple sharings or commitments in SETUP-DIST, with probability at least $1 - \frac{2n}{2^{\kappa/2}}$, all parties take `fail` as output in VERIFICATION.

Proof. Note that if the check passes, by the evaluation binding property of the KZG commitment, for each sharing, the shares held by honest parties must be consistent with the first $t+1$ shares that pass the verification. Therefore, this check equivalently verifies whether the sharings held by honest parties are correct couple sharings and whether the commitments are correctly computed based on the shares held by honest parties.

In the case that at least one party P'_i did not generate valid couple sharings or commitments, the number of λ such that both checks pass is bounded by $2n-1$. The lemma follows from Lemma A.3. □

Single Verification. This step is only invoked when all parties take `fail` as output in VERIFICATION. In this case, we want to pinpoint the parties who deviate from the protocol.

Recall that λ is the challenge generated in VERIFICATION. To check P'_i, parties in the old committee publishes their shares of $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t, [\nu^{(i)}]_t + \lambda[v^{(i)}]_t$ and the associated witnesses. Parties in the new committee publishes their shares of $[\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t, [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t$ and the associated witnesses. For each sharing, all parties use the first $t+1$ shares that pass the verification to reconstruct the whole sharing.

For each $i \in [n]$, all parties compute the commitments

$$\begin{aligned} & \mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t) \\ = \ & \mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t; [\nu^{(i)}]_t) \cdot (\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]_t))^{\lambda}, \end{aligned}$$

and

$$\begin{aligned} & \mathcal{C}om_{\text{KZG}}([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t) \\ = \ & \mathcal{C}om_{\text{KZG}}([\tilde{\mu}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t) \cdot (\mathcal{C}om_{\text{KZG}}([\tilde{u}^{(i)}]_t; [\tilde{v}^{(i)}]_t))^{\lambda}. \end{aligned}$$

Then check if

Protocol 14: VERIFICATION

1. All parties invoke DISTRIBUTION and ACCUSATION-RESPONSE. Let $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t), ([\nu^{(i)}]_t, [\tilde{\nu}^{(i)}]_t)$ denote the random couple sharings generated by $P'_i \in \mathcal{C}'$.
2. All parties in the new committee $\mathcal{C}'$ invoke CHALLENGE to generate a challenge $\lambda \in \mathbb{Z}_p$.
3. Let

$$F(X) = \sum_{i=1}^{n} (\mu^{(i)} + u^{(i)}X)X^{2i-2}, \quad \tilde{F}(X) = \sum_{i=1}^{n} (\tilde{\mu}^{(i)} + \tilde{u}^{(i)}X)X^{2i-2},$$
$$H(X) = \sum_{i=1}^{n} (\nu^{(i)} + v^{(i)}X)X^{2i-2}, \quad \tilde{H}(X) = \sum_{i=1}^{n} (\tilde{\nu}^{(i)} + \tilde{v}^{(i)}X)X^{2i-2}.$$

   Each party P_j in the old committee publishes the j-th shares of $[F(\lambda)]_t, [H(\lambda)]_t$ and the witness $w_j([F(\lambda)]_t; [H(\lambda)]_t)$. Each party P'_j in the new committee publishes the j-th shares of $[\tilde{F}(\lambda)]_t, [\tilde{H}(\lambda)]_t$ and the witness $w_j([\tilde{F}(\lambda)]_t; [\tilde{H}(\lambda)]_t)$.
4. All parties compute the commitment $\mathcal{C}om_{\mathrm{KZG}}([F(\lambda)]_t; [H(\lambda)]_t)$ using

$$\{\mathcal{C}om_{\mathrm{KZG}}([u^{(i)}]_t; [v^{(i)}]_t)\}_{i\in[n]}, \{\mathcal{C}om_{\mathrm{KZG}}([\mu^{(i)}]_t; [\nu^{(i)}]_t)\}_{i\in[n]},$$

   and compute the commitment $\mathcal{C}om_{\mathrm{KZG}}([\tilde{F}(\lambda)]_t; [\tilde{H}(\lambda)]_t)$ using

$$\{\mathcal{C}om_{\mathrm{KZG}}([\tilde{u}^{(i)}]_t; [\tilde{v}^{(i)}]_t)\}_{i\in[n]}, \{\mathcal{C}om_{\mathrm{KZG}}([\tilde{\mu}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t)\}_{i\in[n]}.$$

5. All parties use the first $t+1$ shares that pass the verification to reconstruct the whole sharings $[F(\lambda)]_t, [\tilde{F}(\lambda)]_t, [H(\lambda)]_t, [\tilde{H}(\lambda)]_t$. All parties check if
   (a) $F(\lambda) = \tilde{F}(\lambda)$ and $H(\lambda) = \tilde{H}(\lambda)$,
   (b) $\mathcal{C}om_{\mathrm{KZG}}([F(\lambda)]_t; [H(\lambda)]_t)$ and $\mathcal{C}om_{\mathrm{KZG}}([\tilde{F}(\lambda)]_t; [\tilde{H}(\lambda)]_t)$ are correctly computed.

   If any check fails, all parties take `fail` as output.

1. $\mu^{(i)} + \lambda u^{(i)} = \tilde{\mu}^{(i)} + \lambda \tilde{u}^{(i)}$ and $\nu^{(i)} + \lambda v^{(i)}, \tilde{\nu}^{(i)} + \lambda \tilde{v}^{(i)}$,
2. $\mathcal{C}om_{\mathrm{KZG}}([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$ and
   $\mathcal{C}om_{\mathrm{KZG}}([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t)$ are correctly computed.

If the check fails, P'_i is regarded as a corrupted party. The description of SINGLE-VERI appears in Protocol 15. The communication complexity of each call of SINGLE-VERI is $O(n)$ elements of broadcasting. When all parties behave honestly, there is no need to run SINGLE-VERI. Otherwise, SINGLE-VERI is invoked for each dealer P'_i, resulting in $O(n^2)$ elements of broadcasting in total.

Protocol 15: SINGLE-VERI(P'_i)

1. Each party P_j in the old committee publishes the j-th shares of $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t, [\nu^{(i)}]_t + \lambda[v^{(i)}]_t$ and the witness $w_j([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$. Each party P'_j in the new committee publishes the j-th shares of $[\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t, [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t$ and the witness $w_j([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t)$.
2. All parties compute the commitments $\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$ by
$$\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t; [\nu^{(i)}]_t) \cdot (\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]_t))^{\lambda}$$
and $\mathcal{C}om_{\text{KZG}}([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t)$ by
$$\mathcal{C}om_{\text{KZG}}([\tilde{\mu}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t) \cdot (\mathcal{C}om_{\text{KZG}}([\tilde{u}^{(i)}]_t; [\tilde{v}^{(i)}]_t))^{\lambda}.$$
3. All parties use the first $t+1$ shares that pass the verification to reconstruct the whole sharings $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t, [\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t, [\nu^{(i)}]_t + \lambda[v^{(i)}]_t, [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t$. All parties check if
   (a) $\mu^{(i)} + \lambda u^{(i)} = \tilde{\mu}^{(i)} + \lambda\tilde{u}^{(i)}$ and $\nu^{(i)} + \lambda v^{(i)}, \tilde{\nu}^{(i)} + \lambda\tilde{v}^{(i)}$,
   (b) $\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$ and $\mathcal{C}om_{\text{KZG}}([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t; [\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t)$ are correctly computed.

   If any check fails, P'_i is regarded as a corrupted party.

Lemma C.2. With probability at least $1 - \frac{n}{2^{\kappa/2}}$, all corrupted parties that did not generate valid couple sharings or commitments in SETUP-DIST can be identified in SINGLE-VERI.

Proof. Note that if the check passes, by the evaluation binding property of the KZG commitment, for each sharing, the shares held by honest parties must be consistent with the first $t+1$ shares that pass the verification. Therefore, this check equivalently verifies whether the sharings held by honest parties are correct couple sharings and whether the commitments are correctly computed based on the shares held by honest parties.

In the case that P'_i did not generate valid couple sharings or commitments, the number of λ such that both checks pass is bounded by 1. The lemma follows from the union bound and Lemma A.3. □

Output. For each identified corrupted party P'_i, all parties use all 0 sharings instead of the one distributed by P'_i. Let $\boldsymbol{M}^{\text{T}}$ be a fixed Vandermonde matrix of size $n \times (n-t)$. Then, the old committee $\mathcal{C}$ computes

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\text{T}} = \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\text{T}}$$
$$([\psi^{(1)}]_t, [\psi^{(2)}]_t, \ldots, [\psi^{(n-t)}]_t)^{\text{T}} = \boldsymbol{M}([v^{(1)}]_t, [v^{(2)}]_t, \ldots, [v^{(n)}]_t)^{\text{T}}.$$

The new committee $\mathcal{C}'$ computes

$$\begin{aligned}([\tilde{r}^{(1)}]_t, [\tilde{r}^{(2)}]_t, \ldots, [\tilde{r}^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([\tilde{u}^{(1)}]_t, [\tilde{u}^{(2)}]_t, \ldots, [\tilde{u}^{(n)}]_t)^{\mathrm{T}} \\ ([\tilde{\psi}^{(1)}]_t, [\tilde{\psi}^{(2)}]_t, \ldots, [\tilde{\psi}^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([\tilde{v}^{(1)}]_t, [\tilde{v}^{(2)}]_t, \ldots, [\tilde{v}^{(n)}]_t)^{\mathrm{T}}.\end{aligned}$$

Note that the KZG commitment scheme is linear homomorphism. Therefore, the old committee $\mathcal{C}$ computes the commitments $\{\mathcal{C}om_{\mathrm{KZG}}([r^{(i)}]_t; [\psi^{(i)}]_t)\}_{i\in[n-t]}$ and the witnesses associated with their shares. The new committee $\mathcal{C}'$ computes the commitments $\{\mathcal{C}om_{\mathrm{KZG}}([\tilde{r}^{(i)}]_t; [\tilde{\psi}^{(i)}]_t)\}_{i\in[n-t]}$ and the witnesses associated with their shares.

Finally, output the following $(n-t)$ pairs of couple sharings

$$([r^{(1)}]_t, [\tilde{r}^{(1)}]_t), \ldots, ([r^{(n-t)}]_t, [\tilde{r}^{(n-t)}]_t).$$

According to Lemma A.2, the resulting $(n-t)$ pairs of couple sharings are uniformly random. The description of Output appears in Protocol 16. Note that Output does not require any communication.

Protocol 16: Output

1. For each party $P'_i \in \mathcal{C}'$, if P'_i is identified as a corrupted party, all sharings generated by P'_i are replaced by all 0 sharings.
2. Parties in the old committee $\mathcal{C}$ compute
$$\begin{aligned}([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}} \\ ([\psi^{(1)}]_t, [\psi^{(2)}]_t, \ldots, [\psi^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([v^{(1)}]_t, [v^{(2)}]_t, \ldots, [v^{(n)}]_t)^{\mathrm{T}}.\end{aligned}$$
3. All parties in the old committee compute the commitments $\{\mathcal{C}om_{\mathrm{KZG}}([r^{(i)}]_t; [\psi^{(i)}]_t)\}_{i\in[n-t]}$ and the witnesses associated with their shares.
4. Parties in the new committee $\mathcal{C}'$ compute
$$\begin{aligned}([\tilde{r}^{(1)}]_t, [\tilde{r}^{(2)}]_t, \ldots, [\tilde{r}^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([\tilde{u}^{(1)}]_t, [\tilde{u}^{(2)}]_t, \ldots, [\tilde{u}^{(n)}]_t)^{\mathrm{T}} \\ ([\tilde{\psi}^{(1)}]_t, [\tilde{\psi}^{(2)}]_t, \ldots, [\tilde{\psi}^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([\tilde{v}^{(1)}]_t, [\tilde{v}^{(2)}]_t, \ldots, [\tilde{v}^{(n)}]_t)^{\mathrm{T}}.\end{aligned}$$
5. All parties in the new committee compute the commitments $\{\mathcal{C}om_{\mathrm{KZG}}([\tilde{r}^{(i)}]_t; [\tilde{\psi}^{(i)}]_t)\}_{i\in[n-t]}$ and the witnesses associated with their shares.

Refresh Phase We will maintain the invariance that for each secret s, the old committee $\mathcal{C}$ holds the sharing $[s]_t$. In addition, the old committee holds another random sharing $[z]_t$, which is used to commit the sharing $[s]_t$. The commitment of $[s]_t$, i.e., $\mathcal{C}om_{\mathrm{KZG}}([s]_t; [z]_t)$ has been published. Each party $P_j \in \mathcal{C}$ holds the witness $w_j([s]_t; [z]_t)$.

For each secret sharing $[s]_t$ held by the old committee, one pair of couple sharings $([r]_t, [\tilde{r}]_t)$ generated in the preparation phase is consumed. Let $[\psi]_t$ and $[\tilde{\psi}]_t$ be the random sharings which are used to commit $[r]_t$ and $[\tilde{r}]_t$ respectively. Then the commitments of $[r]_t, [\tilde{r}]_t$, i.e., $\mathcal{C}om_{\text{KZG}}([r]_t; [\psi]_t)$, $\mathcal{C}om_{\text{KZG}}([\tilde{r}]_t; [\tilde{\psi}]_t)$ have been published. Each party $P_j \in \mathcal{C}$ holds the witness $w_j([r]_t; [\psi]_t)$. Each party $P'_j \in \mathcal{C}'$ holds the witness $w_j([\tilde{r}]_t; [\tilde{\psi}]_t)$. A special party $P_{\texttt{king}} \in \mathcal{C}$ is selected and responsible to do the reconstruction. The description of REFRESH appears in Protocol 17. The communication complexity of REFRESH is $O(n)$ elements plus $O(1)$ broadcasting.

Protocol 17: REFRESH

1. All parties in the old committee compute $[s+r]_t = [s]_t + [r]_t$ and $[z+\psi]_t = [z]_t + [\psi]_t$.
2. Each party $P_j \in \mathcal{C}$ computes the witness associated with its share of $[s+r]_t$, i.e., $w_j([s+r]_t; [z+\psi]_t) = w_j([s]_t; [z]_t) w_j([r]_t; [\psi]_t)$.
3. All parties compute the commitment of $[s+r]_t$ by $\mathcal{C}om_{\text{KZG}}([s+r]_t; [z+\psi]_t) = \mathcal{C}om_{\text{KZG}}([s]_t; [z]_t) \mathcal{C}om_{\text{KZG}}([r]_t; [\psi]_t)$.
4. Each party $P_j \in \mathcal{C}$ sends its shares of $[s+r]_t, [z+\psi]_t$ and the associated witness to $P_{\texttt{king}}$.
5. $P_{\texttt{king}}$ verifies the correctness of the received shares and reconstructs $s+r, z+\psi$ using the first $t+1$ shares that pass the verification.
6. $P_{\texttt{king}}$ computes the witness $w_0([s+r]_t; [z+\psi]_t)$. Note that $P_{\texttt{king}}$ holds enough shares to reconstruct the whole sharings $[s+r]_t, [z+\psi]_t$.
7. $P_{\texttt{king}}$ publishes the values $s+r, z+\psi$ and the witness $w_0([s+r]_t; [z+\psi]_t)$.
8. All parties check the correctness of the reconstruction. If the check fails, all parties regard $P_{\texttt{king}}$ as a corrupted party and take $(\texttt{corrupted}, P_{\texttt{king}})$ as output. Otherwise, all parties in the new committee compute $[\tilde{s}]_t = (s+r) - [\tilde{r}]_t, [\tilde{z}]_t = (z+\psi) - [\tilde{\psi}]_t$. The last step is done by regarding $(s+r), (z+\psi)$ as constant sharings, i.e., each party takes $(s+r), (z+\psi)$ as its shares. Since the whole sharings are public, each party $P'_j \in \mathcal{C}'$ can compute $\mathcal{C}om_{\text{KZG}}(s+r; z+\psi)$ and $w_j(s+r; z+\psi)$.
9. Each party $P'_j \in \mathcal{C}'$ computes the witness associated with its share of $[\tilde{s}]_t$, i.e., $w_j([\tilde{s}]_t; [\tilde{z}]_t) = w_j(s+r; z+\psi)/w_j([\tilde{r}]_t; [\tilde{\psi}]_t)$, and the commitment of $[\tilde{s}]_t$, i.e., $\mathcal{C}om_{\text{KZG}}([\tilde{s}]_t; [\tilde{z}]_t) = \mathcal{C}om_{\text{KZG}}(s+r; z+\psi)/\mathcal{C}om_{\text{KZG}}([\tilde{r}]_t; [\tilde{\psi}]_t)$.

The full description of the hand-off phase appears in Protocol 18. We give a brief analysis of the communication complexity.

- Preparation Phase:
  - When all parties behave honestly, the communication complexity is $O(n^2)$ elements plus $O(n)$ elements of broadcasting. On average, each couple sharing requires $O(n)$ elements

plus $O(1)$ elements of broadcasting.

  - When one or more parties behave maliciously, the communication complexity is $O(n^2)$ elements plus $O(n^2)$ elements of broadcasting. On average, each couple sharing requires $O(n)$ elements plus $O(n)$ elements of broadcasting.

- Refresh Phase:
  - When all parties behave honestly, the communication complexity per secret is $O(n)$ elements plus $O(1)$ elements of broadcasting.
  - In the worst case, the communication complexity per secret is $O(n^2)$ elements plus $O(n)$ elements of broadcasting.

In summary, when all parties behave honestly, refreshing each secret requires $O(n)$ elements plus $O(1)$ elements of broadcasting. In the worst case, refreshing each secret requires $O(n^2)$ elements plus $O(n)$ elements of broadcasting.

Protocol 18: HAND-OFF

- **Preparation Phase**
  1. All parties invoke DISTRIBUTION and ACCUSATION-RESPONSE.
  2. All parties invoke VERIFICATION to verify the correctness of the sharings generated in Step 1. If VERIFICATION fails, all parties invoke SINGLE-VERI(P'_i) for all $P'_i \in \mathcal{C}'$.
  3. All parties invoke OUTPUT.
- **Refresh Phase**

  For each secret sharing $[s]_t$ held by the old committee, one pair of random couple sharings $([r]_t, [\tilde{r}]_t)$ is consumed. Repeat the following steps until $[s]_t$ is successfully shared to the new committee.
  1. All parties agree on a special party $P_{\texttt{king}}$ which is not identified as a corrupted party.
  2. All parties invoke REFRESH.

C.2 Setup Phase

At the beginning of the protocol, a trusted third party prepares the public key PK for the KZG commitment scheme. Then parties in the first committee run a distribution protocol with the client to share the secrets.

At a high level, the committee first uses a similar approach to the preparation phase of the hand-off phase to prepare random sharings. For each secret s, one random sharing $[r]_t$ is consumed. The client collects the whole sharing $[r]_t$ and then reconstructs the value r. Finally, the client publishes $s + r$. The committee can compute $[s]_t = s + r - [r]_t$.

Preparing Random Sharings with Polynomial Commitments The descriptions of Setup-Dist, Setup-Acc-Res Setup-Veri, Setup-Single-Veri and Setup-Output appear in Protocol 19, Protocol 20, Protocol 21, Protocol 22 and Protocol 23 respectively. Compared with the preparation phase in the hand-off phase, each dealer only needs to prepare random sharings instead of random couple sharings.

Protocol 19: Setup-Dist

1. Each party P_i randomly samples two sharings $[u^{(i)}]_t$ and $[v^{(i)}]_t$.
2. Each party P_i computes $\mathcal{C}om_{\mathrm{KZG}}([u^{(i)}]_t, [v^{(i)}]_t)$ and publishes the commitment.
3. Each party P_i distributes $[u^{(i)}]_t, [v^{(i)}]_t$ associated with the witnesses to all other parties.
4. All parties verify the validity of their shares and witnesses by invoking $\mathsf{VerifyEval}$ of the KZG commitment scheme.

Protocol 20: Setup-Acc-Res

1. For each party P_j, if the shares and the witness received from P_i are invalid, P_j publishes $(\texttt{accuse}, P_i)$.
2. For each accusation against P_i made by P_j, P_i publishes the shares and the witness which were sent to P_j.
3. For each accusation against P_i made by P_j, all parties check the validity of the shares and the witness published by P_i.
   - If the shares and the witness are invalid, all parties regard P_i as a corrupted party.
   - Otherwise, P_j uses the shares and the witness published by P_i.

Lemma C.3. If at least one party P_i did not generate valid commitments in Setup-Dist, with probability at least $1 - \frac{2n}{2^{\kappa/2}}$, all parties take $\texttt{fail}$ as output in Setup-Veri.

Lemma C.4. With probability at least $1 - \frac{n}{2^{\kappa/2}}$, all corrupted parties that did not generate valid commitments in Setup-Dist can be identified in Setup-Single-Veri.

Fresh Phase After random sharings are prepared, the client acts like the $P_{\texttt{king}}$ in Refresh. For each secret s, a random sharing $[r]_t$ is consumed. Let $[\psi]_t$ denote the random sharing which is used

Protocol 21: SETUP-VERI

1. All parties invoke SETUP-DIST and SETUP-ACC-RES. Let $[\mu^{(i)}]_t, [\nu^{(i)}]_t$ denote the random sharings generated by P_i.
2. All parties invoke CHALLENGE to generate a challenge $\lambda \in \mathbb{Z}_p$.
3. Let
$$F(X) = \sum_{i=1}^{n} (\mu^{(i)} + u^{(i)}X)X^{2i-2}, \quad H(X) = \sum_{i=1}^{n} (\nu^{(i)} + v^{(i)}X)X^{2i-2}.$$
Each party P_j publishes the j-th shares of $[F(\lambda)]_t, [H(\lambda)]_t$ and the witness $w_j([F(\lambda)]_t; [H(\lambda)]_t)$.
4. All parties compute the commitment $\mathcal{C}om_{\text{KZG}}([F(\lambda)]_t; [H(\lambda)]_t)$ using $\{\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]_t)\}_{i\in[n]}, \{\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t; [\nu^{(i)}]_t)\}_{i\in[n]}$.
5. All parties use the first $t+1$ shares that pass the verification to reconstruct the whole sharings $[F(\lambda)]_t, [H(\lambda)]_t$. All parties check if $\mathcal{C}om_{\text{KZG}}([F(\lambda)]_t; [H(\lambda)]_t)$ is correctly computed. If not, all parties take `fail` as output.

Protocol 22: SETUP-SINGLE-VERI(P_i)

1. Each party P_j publishes the j-th shares of $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t, [\nu^{(i)}]_t + \lambda[v^{(i)}]_t$ and the witness $w_j([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$.
2. All parties compute the commitment $\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$ by
$$\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t; [\nu^{(i)}]_t) \cdot (\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]_t))^{\lambda}.$$
3. All parties use the first $t+1$ shares that pass the verification to reconstruct the whole sharings $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t, [\nu^{(i)}]_t + \lambda[v^{(i)}]_t$. All parties check if $\mathcal{C}om_{\text{KZG}}([\mu^{(i)}]_t + \lambda[u^{(i)}]_t; [\nu^{(i)}]_t + \lambda[v^{(i)}]_t)$ is correctly computed. If not, P_i is regarded as a corrupted party.

to compute the commitment of $[r]_t$, i.e., $\mathcal{C}om_{\text{KZG}}([r]_t; [\psi]_t)$. All parties hold their shares of $[r]_t, [\psi]_t$ and the associated witnesses. The description of SETUP-FRESH appears in Protocol 24. The communication complexity of SETUP-FRESH is $O(n)$ elements plus $O(1)$ elements of broadcasting.

The full description of the setup phase appears in Protocol 25. When all parties behave honestly, sharing each secret requires $O(n)$ elements plus $O(1)$ elements of broadcasting. In the worst case,

Protocol 23: Setup-Output

1. For each party P_i, if P_i is identified as a corrupted party, all sharings generated by P_i are replaced by all 0 sharings.
2. All parties compute

$$\begin{aligned}([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}} \\ ([\psi^{(1)}]_t, [\psi^{(2)}]_t, \ldots, [\psi^{(n-t)}]_t)^{\mathrm{T}} &= \boldsymbol{M}([v^{(1)}]_t, [v^{(2)}]_t, \ldots, [v^{(n)}]_t)^{\mathrm{T}}.\end{aligned}$$

3. All parties compute the commitments $\{\mathcal{C}om_{\mathrm{KZG}}([r^{(i)}]_t; [\psi^{(i)}]_t)\}_{i\in[n-t]}$ and the witnesses associated with their shares.

Protocol 24: Setup-Fresh

1. All parties send their shares of $[r]_t, [\psi]_t$ and the associated witnesses to the client.
2. The client verifies the correctness of the received shares and reconstructs r, ψ using the first $t+1$ shares that pass the verification.
3. The client randomly samples z. Then publish $s+r, z+\psi$.
4. All parties compute $[s]_t = (s+r) - [r]_t, [z]_t = (z+\psi) - [\psi]_t$. The last step is done by regarding $(s+r), (z+\psi)$ as constant sharings, i.e., each party takes $(s+r), (z+\psi)$ as its shares. Since the whole sharings are public, each party can compute $\mathcal{C}om_{\mathrm{KZG}}(s+r; z+\psi)$ and $w_j(s+r; z+\psi)$.
5. Each party computes the witness associated with its share of $[s]_t$, i.e., $w_j([s]_t; [z]_t) = w_j(s+r; z+\psi)/w_j([r]_t; [\psi]_t)$, and the commitment of $[s]_t$, i.e., $\mathcal{C}om_{\mathrm{KZG}}([s]_t; [z]_t) = \mathcal{C}om_{\mathrm{KZG}}(s+r; z+\psi)/\mathcal{C}om_{\mathrm{KZG}}([r]_t; [\psi]_t)$.

sharing each secret requires $O(n)$ elements plus $O(n)$ elements of broadcasting.

C.3 Reconstruction Phase

When asking to reconstruct some secret $s^\star$ to a client, all parties in the current committee simply send their shares of $[s^\star]_t$ and the associated witnesses to the client. Then the client can reconstruct the secret using the first $t+1$ shares that pass the verification. The description of Reconstruction appears in Protocol 26. The communication complexity of Reconstruction is $O(n)$ elements.

Protocol 25: SETUP

- **Preparation Phase**
  1. All parties invoke SETUP-DIST and SETUP-ACC-RES.
  2. All parties invoke SETUP-VERI to verify the correctness of the sharings generated in Step 1. If SETUP-VERI fails, all parties invoke SETUP-SINGLE-VERI(P_i) for all P_i.
  3. All parties invoke SETUP-OUTPUT.
- **Fresh Phase**

  For each secret s, one random sharing $[r]_t$ is consumed. All parties invoke SETUP-FRESH.

Protocol 26: RECONSTRUCTION

Suppose $[s^\star]_t$ is the sharing all parties want to reconstruct and Client is the receiver.

1. All parties in the current committee send their shares of $[s^\star]_t$ and the associated witnesses to Client.
2. Client verifies the correctness of the received shares and witnesses. Then reconstruct $s^\star$ using the first $t+1$ shares that pass the verification.

C.4 Full DPSS Protocol

The full construction is presented in Protocol 27. We summarize the communication complexity of each phase as follows:

- Setup Phase: When all parties behave honestly, sharing each secret requires $O(n)$ elements plus $O(1)$ elements of broadcasting. In the worst case, sharing each secret requires $O(n)$ elements plus $O(n)$ elements of broadcasting.
- Hand-off Phase: When all parties behave honestly, refreshing each secret requires $O(n)$ elements plus $O(1)$ elements of broadcasting. In the worst case, refreshing each secret requires $O(n^2)$ elements plus $O(n)$ elements of broadcasting.
- Reconstruction Phase: Reconstructing each secret requires $O(n)$ field elements.

We present the proof of robustness and the proof of security of our DPSS construction in Appendix D.

Protocol 27: MAIN

- **Setup Phase**
    1. A trusted party invokes $\mathsf{Setup}(1^\kappa, t)$ of the KZG commitment scheme and publishes the public key PK.
    2. The first committee and the client invoke SETUP to share the secret values.
    3. All parties only keep their shares of the secrets and the associated witnesses and erase anything else.
- **Hand-off Phase**
    1. All parties invoke HAND-OFF to pass the sharings of the secrets from the old committee to the new committee.
    2. Parties in the old committee erases all their shares and witnesses. Parties in the new committee only keep their shares of the secrets and the associated witnesses and erase anything else.
- **Reconstruction Phase**

  Let $s^\star$ be the secret all parties need to reconstruct and Client be the receiver. All parties invoke RECONSTRUCTION on $[s^\star]_t$ and Client.

C.5 Extension to Various Thresholds

We note that our construction can be easily extended to handle various sizes of committees with the only requirement that each committee satisfies honest majority. At a high-level, we simply use secret sharings of different thresholds for different committees.

Specifically, let n, t denote the number of parties and the threshold for the old committee, and, n', t', for the new committee. Now the couple sharings will be two sharings of the same value where one is a degree-t sharing and the other one is a degree-t' sharing.

When preparing couple sharings in the preparation phase, each party in the new committee will generate a pair of random couple sharings and distribute the degree-t sharing to the old committee and the degree-t' sharing to the new committee. The verification step processes as before except that the old committee uses the threshold t and the new committee uses the threshold t'. The refresh phase remains unchanged.

The security can be shown in a similar way as that of Protocol 27.

D DPSS Security Proof

D.1 DPSS Robustness Proof

To show the robustness, it is sufficient to prove the following three lemmas.

Lemma D.1. With overwhelming probability, after the setup phase, there exists a fixed secret for each sharing. Especially, if the client is honest, then the secret is the same as the one chosen by the client.

Proof. According to Lemma C.3, all parties can detect misbehaviors in Setup-Veri with overwhelming probability. Then in Setup-Single-Veri, according to Lemma C.4 and the union bound, all corrupted parties that did not generate valid commitments in Setup-Dist can be identified. The reason for using the union bound is because we use the same challenge string λ in Setup-Veri and Setup-Single-Veri.

Therefore, in Setup-Output, for each $P_i \in \mathcal{C}$ which is not identified as a corrupted party, with overwhelming probability,

1. for each sharing distributed by P_i, the shares held by honest parties are consistent.
2. honest parties hold valid witnesses associated with their shares and the KZG commitments published by P_i are correctly computed.

Note that, from the shares held by honest parties, we can compute the shares and the associated witnesses that corrupted parties should hold. According to the evaluation binding property of the KZG commitment scheme, a corrupted party cannot generate a different share with a valid witness.

Therefore, with overwhelming probability, for each sharing $[r]_t$ output by Setup-Output, the secret r can be determined by the shares held by honest parties, and, corrupted parties cannot generate different shares with valid witnesses. Thus, in Setup-Fresh, the client is able to compute the correct secret r using the first $t+1$ shares that pass the verification. Suppose the client publishes $s+r$. Since $[r]_t$ is correctly shared among the committee $\mathcal{C}$, all parties in $\mathcal{C}$ can compute the sharing $[s]_t = (s+r)-[r]_t$. According to the correctness of $[r]_t$, (1) honest parties in $\mathcal{C}$ hold consistent shares of $[s]_t$ with valid witnesses, (2) the KZG commitment of $[s]_t$ is correctly computed. According to the evaluation binding property of the KZG commitment scheme, a corrupted party in $\mathcal{C}$ is unable to provide a wrong share of $[s]_t$ with a valid witness.

The secret of the sharing $[s]_t = s + r - [r]_t$ is s, which is a fixed value and determined by the shares held by honest parties. In particular, if the client is honest, then s remains the same as the choice of the client. □

Lemma D.2. With overwhelming probability, after each of the hand-off phase, for each secret, all honest parties in the new committee $\mathcal{C}'$ receive the correct shares of the original secret. For any set $\mathcal{T} \subset \mathcal{C}'$, the shares provided by parties in $\mathcal{T}$ can only be reconstructed (by an honest reconstructor) to either the original secret or $\perp$, where $\perp$ means a failure in the reconstruction. In particular, if $\mathcal{T}$ contains at least $t+1$ honest parties, then the shares provided by parties in $\mathcal{T}$ can always be reconstructed (by an honest reconstructor) to the original secret.

Proof. According to Lemma C.1, all parties can detect misbehaviors in Verification with overwhelming probability. Then in Single-Veri, according to Lemma C.2 and the union bound, all corrupted parties that did not generate valid couple sharings or commitments can be identified. The reason for using the union bound is because we use the same challenge string λ in Verification and Single-Veri.

Therefore, in Output, for each $P'_i \in \mathcal{C}'$ which is not identified as a corrupted party, with overwhelming probability,

1. for each sharing distributed by P'_i, the shares held by honest parties are consistent.
2. the secrets of the two sharings in a pair of couple sharings are the same.
3. honest parties hold valid witnesses associated with their shares and the KZG commitments published by P'_i are correctly computed.

Note that, from the shares held by honest parties, we can compute the shares and the associated witnesses that corrupted parties should hold. According to the evaluation binding property of the KZG commitment scheme, a corrupted party cannot generate a different share with a valid witness.

Assume the robustness holds for the old committee $\mathcal{C}$. Then, in REFRESH, a corrupted party cannot provide a wrong share of $[s+r]_t$ with a valid witness to $P_{\texttt{king}}$. Since $P_{\texttt{king}}$ receives at least $t+1$ correct shares with valid witnesses from honest parties, $P_{\texttt{king}}$ can always reconstruct the sharing $[s+r]_t$. In addition, according to the evaluation binding property of the KZG commitment scheme again, a corrupted $P_{\texttt{king}}$ cannot generate a value different from $s+r$ with a valid witness. Since $[\tilde{r}]_t$ is correctly shared among the new committee $\mathcal{C}'$ and $r = \tilde{r}$, all parties in $\mathcal{C}'$ can compute the sharing $[\tilde{s}]_t = (s+r) - [\tilde{r}]_t$ such that $\tilde{s} = s$. According to the correctness of $[\tilde{r}]_t$, (1) honest parties in $\mathcal{C}'$ hold consistent shares of $[\tilde{s}]_t$ with valid witnesses, (2) the secret is $\tilde{s} = s$ and (3) the KZG commitment of $[\tilde{s}]_t$ is correctly computed. According to the evaluation binding property of the KZG commitment scheme, a corrupted party in $\mathcal{C}'$ is unable to provide a wrong share of $[s]_t$ with a valid witness.

Therefore, for any subset $\mathcal{T} \subset \mathcal{C}'$, either the shares provided by parties in $\mathcal{T}$ contain at least $t+1$ valid shares, which can be used to reconstruct the correct secret, or contain at most t valid shares, which leads to an abort of the reconstruction. □

Lemma D.3. With overwhelming probability, the client in RECONSTRUCTION can always reconstruct the correct secret $s^\star$.

Proof. Note that the current committee contains at least $t+1$ honest parties. The statement follows from Lemma D.2. □

D.2 DPSS Secrecy Proof

We will first construct a simulator $\mathcal{S}$ and then show that the transcript of the interaction between $\mathcal{S}$ and the adversary $\mathcal{A}$ is indistinguishable from the real execution.

Simulating the Setup Phase. The simulator $\mathcal{S}$ first prepares the public key. $\mathcal{S}$ invokes $\mathsf{Setup}(1^\kappa, t)$ and receives two groups $\mathbb{G}, \mathbb{G}_T$ of prime order p and a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

$\mathcal{S}$ samples a random generator $g \in \mathbb{G}$. Then randomly sample $\alpha, \beta \in \mathbb{Z}_p$. Let $h = g^\beta$. The public key PK is set to be $(\mathbb{G}, \mathbb{G}_T, e, g, g^\alpha, \ldots, g^{\alpha^t}, h, h^\alpha, \ldots, h^{\alpha^t})$.

Simulating SETUP-DIST. Recall that a sharing $[s]_t$ is also used to denote its corresponding polynomial. We use the notation $[s]_t(x)$ to denote the value at the point x.

For each honest party P_i, the simulator $\mathcal{S}$ chooses two random elements for each corrupted party as its shares of $[u^{(i)}]_t$ and $[v^{(i)}]_t$. Recall that $h = g^\beta$. $\mathcal{S}$ randomly samples a sharing as $[u^{(i)}]_t + \beta[v^{(i)}]_t$ such that the shares held by corrupted parties match the shares of $[u^{(i)}]_t$ and $[v^{(i)}]_t$ chosen for corrupted parties. Then $\mathcal{S}$ computes the commitment

$$\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]) = g^{[u^{(i)}]_t(\alpha)} h^{[v^{(i)}]_t(\alpha)} = g^{([u^{(i)}]_t + \beta[v^{(i)}]_t)(\alpha)}.$$

For each corrupted party P_j, the witness associated with its share $[u^{(i)}]_t(j)$ can be computed by

$$\begin{aligned} w_j([u^{(i)}]_t; [v^{(i)}]_t) &= g^{\frac{[u^{(i)}]_t(\alpha)-[u^{(i)}]_t(j)}{\alpha-j}} h^{\frac{[v^{(i)}]_t(\alpha)-[v^{(i)}]_t(j)}{\alpha-j}} \\ &= g^{\frac{[u^{(i)}]_t(\alpha)+\beta[v^{(i)}]_t(\alpha)-[u^{(i)}]_t(j)-\beta[v^{(i)}]_t(j)}{\alpha-j}} \\ &= g^{\frac{([u^{(i)}]_t+\beta[v^{(i)}]_t)(\alpha)-([u^{(i)}]_t+\beta[v^{(i)}]_t)(j)}{\alpha-j}} \end{aligned}$$

The simulator $\mathcal{S}$ faithfully follows the rest of the steps in Setup-Dist.

Simulating Setup-Acc-Res. Note that an honest party never accuses another honest party. Since all shares sent from honest parties to corrupted parties are generated explicitly, $\mathcal{S}$ simply follows the protocol.

Simulating Setup-Veri *and* Setup-Single-Veri. In the first step, $\mathcal{S}$ simulates Setup-Dist and Setup-Acc-Res in the same way as described above. $\mathcal{S}$ faithfully follows Challenge when generating a challenge $\lambda \in \mathbb{Z}_p$.

For each honest party P_i, $\mathcal{S}$ randomly samples a sharing as $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t$ such that the shares held by corrupted parties match the shares of $[u^{(i)}]_t$ and $[\mu^{(i)}]_t$ sent to corrupted parties. Then $[\nu^{(i)}]_t + \lambda[v^{(i)}]_t$ is set to be

$$\beta^{-1}(\lambda([u^{(i)}]_t + \beta[v^{(i)}]_t) + ([\mu^{(i)}]_t + \beta[\nu^{(i)}]_t) - ([\mu^{(i)}]_t + \lambda[u^{(i)}]_t)).$$

Note that the following two sharings $[u^{(i)}]_t + \beta[v^{(i)}]_t, [\mu^{(i)}]_t + \beta[\nu^{(i)}]_t$ have been explicitly generated when $\mathcal{S}$ simulates Setup-Dist.

The simulator $\mathcal{S}$ faithfully follows the rest of the steps in Setup-Veri and Setup-Single-Veri.

Simulating Setup-Output. Since Setup-Output does not need communication, $\mathcal{S}$ simply follows the protocol. Note that for each pair of sharings $([r^{(i)}]_t, [\psi^{(i)}]_t)$, $\mathcal{S}$ can compute the whole sharing $[r^{(i)}]_t + \beta[\psi^{(i)}]_t$ using $\{[u^{(j)}]_t + \beta[v^{(j)}]_t\}_{j\in[n]}$.

Simulating Setup-Fresh. Depending on whether the client is honest or corrupted, there are two cases.

- If the client is honest, $\mathcal{S}$ chooses two random elements as $s + r, z + \psi$ and publishes them on behalf of the client.

- If the client is corrupted, $\mathcal{S}$ randomly samples a sharing as $[r]_t$ based on the shares held by corrupted parties, and computes $[\psi]_t = \beta^{-1}(([r]_t + \beta[\psi]_t) - [r]_t)$. $\mathcal{S}$ faithfully sends the shares and witnesses of honest parties to the client. After receiving $s + r, z + \psi$ from the client, $\mathcal{S}$ computes the secret s from $s + r, r$ and sends the secret to $\mathsf{Ideal}_{\text{safe}}$.

Finally, $\mathcal{S}$ computes the sharing $[s]_t + \beta[z]_t := ((s + r) + \beta(z + \psi)) - ([r]_t + \beta[\psi]_t)$.

Handling Adaptive Corruptions. Suppose P_j is corrupted by $\mathcal{A}$. The simulator $\mathcal{S}$ needs to prepare the sharings dealt by P_j and the messages sent from other honest parties to P_j.

We first prepare the sharings dealt by P_j. Based on the shares held by corrupted parties, $\mathcal{S}$ randomly samples a sharing as $[u^{(j)}]_t$. Then $[v^{(j)}]_t$ is set to be $\beta^{-1}(([u^{(j)}]_t + \beta[v^{(j)}]_t) - [u^{(j)}]_t)$. Regarding the sharings $[\mu^{(j)}]_t, [\nu^{(j)}]_t$, there are two cases.

- If P_j is corrupted before the challenge λ is generated in Setup-Veri, $[\mu^{(j)}]_t$ and $[\nu^{(j)}]_t$ can be prepared in the same way.

- Otherwise, $[\mu^{(j)}]_t, [\nu^{(j)}]_t$ are set to be $([\mu^{(j)}]_t+\lambda[u^{(j)}]_t)-\lambda\cdot[u^{(j)}]_t$ and $([\nu^{(j)}]_t+\lambda[v^{(j)}]_t)-\lambda\cdot[v^{(j)}]_t$ respectively.

Then, we prepare the messages sent from other honest parties to P_j. Recall that $[r^{(1)}]_t(j), \ldots, [r^{(n-t)}]_t(j)$ are the j-th shares of $[r^{(1)}]_t, \ldots, [r^{(n-t)}]_t$ output by SETUP-OUTPUT. For each $i \in [n-t]$, if $[r^{(i)}]_t(j)$ has not been determined, $\mathcal{S}$ randomly samples an element as $[r^{(i)}]_t(j)$ (Note that if $[r^{(i)}]_t$ is used to share a secret of a corrupted client, then $[r^{(i)}]_t(j)$ may be explicitly generated depending on the time when P_j is corrupted by $\mathcal{A}$).

Let $\mathcal{H} \subseteq \mathcal{C}$ be a set of $(n-t)$ honest parties and $\boldsymbol{M}_{\mathcal{H}}$ denote the submatrix containing the columns with indices $i \in \mathcal{H}$. According to the property of the Vandermonde matrix, $\boldsymbol{M}_{\mathcal{H}}$ is invertible. There is a one-to-one map from the j-th shares of $[r^{(1)}]_t, \ldots, [r^{(n-t)}]_t$ to the j-th shares of $\{[u^{(i)}]_t\}_{P_i \in \mathcal{H}}$. For each honest party $P_i \notin \mathcal{H}$, $\mathcal{S}$ randomly samples an element as the j-th share of $[u^{(i)}]_t$. Then, the $j-$th shares of $\{[u^{(i)}]_t\}_{P_i \in \mathcal{H}}$ can be computed from the j-th shares of $\{[r^{(i)}]_t\}_{i \in [n-t]}$ and $\{[u^{(i)}]_t\}_{P_i \notin \mathcal{H}}$.

Regarding the j-th shares of $[\mu^{(i)}]_t, [\nu^{(i)}]_t$ dealt by an honest party P_i, there are two cases.

- If P_j is corrupted before the challenge λ is generated in SETUP-VERI, for each honest party P_i, the j-th share of $[\mu^{(i)}]_t$ is set to be a uniform element and the j-th share of $[\nu^{(i)}]_t$ is set to be
$$\beta^{-1}(([\mu^{(i)}]_t + \beta[\nu^{(i)}]_t)(j) - [\mu^{(i)}]_t(j)).$$
- Otherwise, the j-th shares of $[\mu^{(i)}]_t, [\nu^{(i)}]_t$ are set to be $([\mu^{(i)}]_t + \lambda[u^{(i)}]_t)(j) - \lambda \cdot [u^{(i)}]_t(j)$ and $([\nu^{(i)}]_t + \lambda[v^{(i)}]_t)(j) - \lambda \cdot [v^{(i)}]_t(j)$ respectively.

Once the j-th shares are prepared, the associated witnesses can be computed in the same way as those for corrupted parties.

Simulating the Hand-off Phase.

We will maintain the invariance that

1. for each sharing $[s]_t$, the simulator $\mathcal{S}$ learns the shares that should be held by corrupted parties.
2. for each commitment $\mathcal{C}om_{\text{KZG}}([s]_t; [z]_t)$, the simulator $\mathcal{S}$ knows the whole sharing $[s]_t + \beta[z]_t$.

It is clear that this invariance holds in the first call of the hand-off phase.

Simulating DISTRIBUTION. DISTRIBUTION is simulated in a similar way to SETUP-DIST.

Concretely, for each honest party $P'_i \in \mathcal{C}'$, the simulator $\mathcal{S}$ chooses two random elements for each corrupted party in the old committee as its shares of $[u^{(i)}]_t$ and $[v^{(i)}]_t$ and two random elements for each corrupted party in the new committee as its shares of $[\tilde{u}^{(i)}]_t$ and $[\tilde{v}^{(i)}]_t$. Recall that $h = g^{\beta}$. $\mathcal{S}$ randomly samples a value as $u^{(i)} + \beta v^{(i)} = \tilde{u}^{(i)} + \beta\tilde{v}^{(i)}$. Based on this value and the shares of corrupted parties, $\mathcal{S}$ randomly samples two sharings as $[u^{(i)}]_t + \beta[v^{(i)}]_t$ and $[\tilde{u}^{(i)}] + \beta[\tilde{v}^{(i)}]_t$.

$\mathcal{S}$ computes the commitments $\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]_t), \mathcal{C}om_{\text{KZG}}([\tilde{u}^{(i)}]_t; [\tilde{v}^{(i)}]_t)$ and the witnesses associated with the shares of corrupted parties in the same way as that when simulating SETUP-DIST.

The simulator $\mathcal{S}$ faithfully follows the rest of the steps in DISTRIBUTION.

Simulating ACCUSATION-RESPONSE. Note that an honest party never accuses another honest party. Since all shares sent from honest parties to corrupted parties are generated explicitly, $\mathcal{S}$ simply follows the protocol.

Simulating Verification *and* Single-Veri. In the first step, $\mathcal{S}$ simulates Distribution and Accusation-Response in the same way as described above. $\mathcal{S}$ faithfully follows Challenge when generating a challenge $\lambda \in \mathbb{Z}_p$.

For each honest party P_i, $\mathcal{S}$ randomly samples a value as $\mu^{(i)} + \lambda u^{(i)} = \tilde{\mu}^{(i)} + \lambda \tilde{u}^{(i)}$. Based on this value and the shares of corrupted parties, $\mathcal{S}$ randomly samples two sharings as $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t$ and $[\tilde{\mu}^{(i)}] + \lambda[\tilde{u}^{(i)}]_t$. Then $[\nu^{(i)}]_t + \lambda[v^{(i)}]_t$ and $[\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t$ are set to be

$$\beta^{-1}(\lambda([u^{(i)}]_t + \beta[v^{(i)}]_t) + ([\mu^{(i)}]_t + \beta[\nu^{(i)}]_t) - ([\mu^{(i)}]_t + \lambda[u^{(i)}]_t)),$$
$$\beta^{-1}(\lambda([\tilde{u}^{(i)}]_t + \beta[\tilde{v}^{(i)}]_t) + ([\tilde{\mu}^{(i)}]_t + \beta[\tilde{\nu}^{(i)}]_t) - ([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t))$$

respectively. Note that the following sharings $[u^{(i)}]_t+\beta[v^{(i)}]_t$, $[\mu^{(i)}]_t+\beta[\nu^{(i)}]_t$ and $[\tilde{u}^{(i)}]_t+\beta[\tilde{v}^{(i)}]_t$, $[\tilde{\mu}^{(i)}]_t+\beta[\tilde{\nu}^{(i)}]_t$ have been explicitly generated when $\mathcal{S}$ simulates Distribution.

The simulator $\mathcal{S}$ faithfully follows the rest of the steps in Verification and Single-Veri.

Simulating Output. Since Output does not need communication, $\mathcal{S}$ simply follows the protocol. Note that for each $i \in [n-t]$, $\mathcal{S}$ can compute the whole sharings $[r^{(i)}]_t + \beta[\psi^{(i)}]_t$ and $[\tilde{r}^{(i)}]_t + \beta[\tilde{\psi}^{(i)}]_t$ using $\{[u^{(j)}]_t + \beta[v^{(j)}]_t\}_{j\in[n]}$ and $\{[\tilde{u}^{(j)}]_t + \beta[\tilde{v}^{(j)}]_t\}_{j\in[n]}$ respectively.

Simulating Refresh. Based on the shares of corrupted parties, $\mathcal{S}$ first chooses a random sharing as $[s+r]_t = [s]_t + [r]_t$. Then, the sharing $[z+\psi]_t$ is set to be

$$\beta^{-1}(([s]_t + \beta[z]_t) + ([r]_t + \beta[\psi]_t) - ([s]_t + [r]_t)).$$

According to the invariance, $\mathcal{S}$ knows the sharing $[s]_t + \beta[z]_t$. $\mathcal{S}$ also learns $[r]_t + \beta[\psi]_t$ when simulating Output.

Since the whole sharings $[s+r]_t$ and $[z+\psi]_t$ are known to $\mathcal{S}$, $\mathcal{S}$ computes the witnesses associated with the shares of honest parties. $\mathcal{S}$ faithfully follows the rest of the protocol.

Finally, $\mathcal{S}$ computes the sharing $[\tilde{s}] + \beta[\tilde{z}] := ((s+r) + \beta(z+\psi)) - ([\tilde{r}]_t + \beta[\tilde{\psi}]_t)$.

Handling Adaptive Corruptions. Depending on which committee the party corrupted by $\mathcal{A}$ belongs to, there are two cases.

- Suppose P'_j in the new committee is corrupted. The simulator $\mathcal{S}$ needs to prepare the shares dealt by P'_j and the messages sent from other honest parties $P'_i \in \mathcal{C}' \backslash \{P'_j\}$.

  For each honest party $P'_i \in \mathcal{C}' \backslash \{P'_j\}$, $\mathcal{S}$ randomly samples an element as the j-th share of $[\tilde{u}^{(i)}]_t$. The j-th share of $[\tilde{v}^{(i)}]_t$ is set to be $\beta^{-1}(([\tilde{u}^{(i)}]_t + \beta[\tilde{v}^{(i)}]_t)(j) - [\tilde{u}^{(i)}]_t(j))$. The witness can be in the same way as that for a corrupted party. For P'_j, based on the shares held by corrupted parties, $\mathcal{S}$ randomly samples a pair of couple sharings $([u^{(j)}]_t, [\tilde{u}^{(j)}]_t)$. Then $[v^{(j)}]_t$ and $[\tilde{v}^{(j)}]_t$ are set to be

  $$\beta^{-1}(([u^{(j)}]_t + \beta[v^{(j)}]_t) - [u^{(j)}]_t)$$
  $$\beta^{-1}(([\tilde{u}^{(j)}]_t + \beta[\tilde{v}^{(j)}]_t) - [\tilde{u}^{(j)}]_t)$$

  respectively.

  1. If P'_j is corrupted before the challenge λ is generated in Verification, $([\mu^{(j)}]_t, [\tilde{\mu}^{(j)}]_t)$, $([\nu^{(j)}]_t, [\tilde{\nu}^{(j)}]_t)$, and the j-th shares of $[\tilde{\mu}^{(i)}]_t, [\tilde{\nu}^{(i)}]_t$ with the witnesses for all honest parties $P'_i \in \mathcal{C}' \backslash \{P'_j\}$ can be prepared in the same way.

2. If P'_j is corrupted after the challenge λ is generated, $([\mu^{(j)}]_t, [\tilde{\mu}^{(j)}]_t)$ and $([\nu^{(j)}]_t, [\tilde{\nu}^{(j)}]_t)$ are set to be

$$([\mu^{(j)}]_t + \lambda[u^{(j)}]_t, [\tilde{\mu}^{(j)}]_t + \lambda[\tilde{u}^{(j)}]_t) - \lambda \cdot ([u^{(j)}]_t, [\tilde{u}^{(j)}]_t)$$
$$([\nu^{(j)}]_t + \lambda[v^{(j)}]_t, [\tilde{\nu}^{(j)}]_t + \lambda[\tilde{v}^{(j)}]_t) - \lambda \cdot ([v^{(j)}]_t, [\tilde{v}^{(j)}]_t)$$

respectively. For each honest party $P'_i \in \mathcal{C}' \backslash \{P'_j\}$, the j-th shares of $[\tilde{\mu}^{(i)}]_t$ and $[\tilde{\nu}^{(i)}]_t$ are set to be

$$([\tilde{\mu}^{(i)}]_t + \lambda[\tilde{u}^{(i)}]_t)(j) - \lambda \cdot [\tilde{u}^{(i)}]_t(j)$$
$$([\tilde{\nu}^{(i)}]_t + \lambda[\tilde{v}^{(i)}]_t)(j) - \lambda \cdot [\tilde{v}^{(i)}]_t(j)$$

respectively. The witness can be computed in the same way as that for a corrupted party.

- Suppose P_j in the old committee is corrupted. The simulator $\mathcal{S}$ needs to prepare the j-th share of each sharing $[s]_t$ and the messages sent from honest parties.

  For each sharing $[s]_t$, let $[z]_t$ denote the random sharing which is used to commit $[s]_t$.

  - If the old committee is the first committee, and $[s]_t$ is dealt by a corrupted client, $\mathcal{S}$ does nothing. Recall that the whole sharing $[s]_t, [z]_t$ were explicitly generated when simulating Setup-Fresh.
  - Otherwise, $\mathcal{S}$ randomly samples an element as the j-th share of $[s]_t$. The j-th share of $[z]_t$ is computed by $\beta^{-1}(([s]_t + \beta[z]_t)(j) - [s]_t(j))$. The witness can be computed by

  $$w_j([s]_t; [z]_t) = g^{\frac{([s]_t+\beta[z]_t)(\alpha) - ([s]_t+\beta[z]_t)(j)}{\alpha - j}}.$$

  Then, we prepare the messages sent from honest parties to P_j. Recall that $[r^{(1)}]_t(j), \ldots, [r^{(n-t)}]_t(j)$ are the j-th shares of $[r^{(1)}]_t, \ldots, [r^{(n-t)}]_t$ output by Output. For each $i \in [n-t]$, if $[r^{(i)}]_t(j)$ has not been determined, $\mathcal{S}$ randomly samples an element as $[r^{(i)}]_t(j)$ (Note that if $[r^{(i)}]_t$ has been used in Refresh for $[s^{(i)}]_t$, since the whole sharing $[s^{(i)} + r^{(i)}]_t$ was generated when simulating Refresh and the j-th share of $[s^{(i)}]_t$ has been prepared as above, the j-th share of $[r^{(i)}]_t$ is determined).

  Let $\mathcal{H} \subseteq \mathcal{C}'$ be a set of $(n-t)$ honest parties and $\boldsymbol{M}_{\mathcal{H}}$ denote the submatrix containing the columns with indices $i \in \mathcal{H}$. According to the property of the Vandermonde matrix, $\boldsymbol{M}_{\mathcal{H}}$ is invertible. There is a one-to-one map from the j-th shares of $[r^{(1)}]_t, \ldots, [r^{(n-t)}]_t$ to the j-th shares of $\{[u^{(i)}]_t\}_{P'_i \in \mathcal{H}}$. For each honest party $P'_i \notin \mathcal{H}$, $\mathcal{S}$ randomly samples an element as the j-th share of $[u^{(i)}]_t$. Then, the $j-$th shares of $\{[u^{(i)}]_t\}_{P'_i \in \mathcal{H}}$ can be computed from the j-th shares of $\{[r^{(i)}]_t\}_{i \in [n-t]}$ and $\{[u^{(i)}]_t\}_{P'_i \notin \mathcal{H}}$.

  Regarding the j-th shares of $[\mu^{(i)}]_t, [\nu^{(i)}]_t$ dealt by an honest party P'_i, there are two cases.

  - If P_j is corrupted before the challenge λ is generated in Verification, for each honest party P'_i, the j-th share of $[\mu^{(i)}]_t$ is set to be a uniform element and the j-th share of $[\nu^{(i)}]_t$ is set to be
  $$\beta^{-1}(([\mu^{(i)}]_t + \beta[\nu^{(i)}]_t)(j) - [\mu^{(i)}]_t(j)).$$

- Otherwise, the j-th shares of $[\mu^{(i)}]_t, [\nu^{(i)}]_t$ are set to be $([\mu^{(i)}]_t + \lambda[u^{(i)}]_t)(j) - \lambda \cdot [u^{(i)}]_t(j)$ and $([\nu^{(i)}]_t + \lambda[v^{(i)}]_t)(j) - \lambda \cdot [v^{(i)}]_t(j)$ respectively.

Once the j-th shares are prepared, the associated witnesses can be computed in the same way as those for corrupted parties.

Simulating the Reconstruction Phase. Note that if the receiver Client is honest, there is no need to send any message to the adversary. $\mathcal{S}$ does nothing. Now assume that Client is corrupted. Suppose $[s^\star]_t$ is the sharing all parties want to reconstruct. In addition, let $[z^\star]_t$ be the sharing used to compute the commitment of $[s^\star]_t$, i.e., $\mathcal{C}om_{\mathrm{KZG}}([s^\star]_t; [z^\star]_t)$. By the invariance, $\mathcal{S}$ learns the whole sharing $[s^\star]_t + \beta[z^\star]_t$.

$\mathcal{S}$ first queries the secret $s^\star$ from $\mathsf{Ideal}_{\mathrm{safe}}$.

- If the current committee has already generated the whole sharing $[s^\star]_t, [z^\star]_t$, $\mathcal{S}$ does nothing. It corresponds to the following two possible scenarios.
  - If $s^\star$ is dealt by a corrupted client and the current committee is the first committee, the whole sharings $[s^\star]_t, [z^\star]_t$ were explicitly generated when simulating Setup-Fresh.
  - If $s^\star$ was just reconstructed to a corrupted client and $[s^\star]_t$ has not been refreshed yet, the whole sharings $[s^\star]_t, [z^\star]_t$ were explicitly generated when simulating the last call of Reconstruction.
- Otherwise, based on the shares of $[s^\star]_t$ held by corrupted parties and the secret $s^\star$, $\mathcal{S}$ randomly samples a sharing as $[s^\star]_t$. Then set $[z^\star]_t$ to be $\beta^{-1}(([s^\star]_t + \beta[z^\star]_t) - [s^\star]_t)$.

Since the whole sharings are prepared, $\mathcal{S}$ computes the witness for each honest party. Finally, $\mathcal{S}$ sends the shares of honest parties with witnesses to Client.

Proving indistinguishability. We show that any PPT adversary $\mathcal{A}$ cannot distinguish the view when interacting with the simulator $\mathcal{S}$ constructed above from the view in the real world. Considering the following hybrids.

Hybrid$_0$: Execution in the real world. The simulator controls all honest parties and follows the protocol.

Hybrid$_1$: The simulator $\mathcal{S}$ prepares the public key PK and stores α, β. The rest of the steps remain the same as **Hybrid$_0$**. The distribution is identical to **Hybrid$_0$**.

Hybrid$_2$: In this hybrid, $\mathcal{S}$ uses the shares of honest parties to recover the secrets dealt by corrupted clients and sends them to $\mathsf{Ideal}_{\mathrm{safe}}$. The distribution is identical to **Hybrid$_1$**.

Hybrid$_3$: In Setup-Dist, $\mathcal{S}$ does not generate the whole sharings $[u^{(i)}]_t, [v^{(i)}]_t$ for each honest party P_i. Instead, $\mathcal{S}$ simulates Setup-Dist, Setup-Acc-Res, Setup-Veri and Setup-Single-Veri in the way we described above. In Setup-Output, $\mathcal{S}$ generates a random sharing $[u^{(i)}]_t$ based on the shares held by corrupted parties.

Recall that $\mathcal{S}$ has generated the whole sharing $[u^{(i)}]_t + \beta[v^{(i)}]_t$ when simulating Setup-Dist. Therefore, $[v^{(i)}]_t$ is set to be $\beta^{-1}(([u^{(i)}]_t + \beta[v^{(i)}]_t) - [u^{(i)}]_t)$.

As for $[\mu^{(i)}]_t$ and $[\nu^{(i)}]_t$, recall that $\mathcal{S}$ has generated the whole sharing $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t$ and $[\nu^{(i)}]_t + \lambda[v^{(i)}]_t$. Therefore, $[\mu^{(i)}]_t$ and $[\nu^{(i)}]_t$ are set to be

$$([\mu^{(i)}]_t + \lambda[u^{(i)}]_t) - \lambda \cdot [u^{(i)}]_t$$
$$([\nu^{(i)}]_t + \lambda[v^{(i)}]_t) - \lambda \cdot [v^{(i)}]_t$$

respectively.

Regarding adaptive corruption, if P_j is corrupted by $\mathcal{A}$ before Setup-Output, the view of P_j is prepared by $\mathcal{S}$ in the way we described above.

Note that for each honest party P_i, the sharing $[\mu^{(i)}]_t$ is prepared as a random mask to protect the secrecy of $[u^{(i)}]_t$. Therefore, $[\mu^{(i)}]_t + \lambda[u^{(i)}]_t$ is a random sharing. $\mathcal{S}$ perfectly simulates the behaviors of honest parties. The distribution is identical to $\mathbf{Hybrid_2}$.

$\mathbf{Hybrid_4}$: Similarly to $\mathbf{Hybrid_3}$, $\mathcal{S}$ simulates Distribution, Accusation-Response, Verification and Single-Veri in the way we described above. The generation of the whole sharings $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t), ([v^{(i)}]_t, [\tilde{v}^{(i)}]_t)$ for each honest party P'_i is postponed to Output. If $P_j \in \mathcal{C} \bigcup \mathcal{C}'$ is corrupted by $\mathcal{A}$ before Output, the view of P_j is prepared by $\mathcal{S}$ in the way we described above. The distribution is identical to $\mathbf{Hybrid_3}$.

$\mathbf{Hybrid_5}$: In Setup-Output, $\mathcal{S}$ changes its strategy of generating $[u^{(i)}]_t, [v^{(i)}]_t$ for each honest party P_i. Recall that all parties compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}}$$

in Setup-Output. Let $\mathcal{H} \subseteq \mathcal{C}$ be a set of $(n-t)$ honest parties and $\boldsymbol{M}_{\mathcal{H}}$ denote the submatrix containing the columns with indices $i \in \mathcal{H}$. By the property of the Vandermonde matrix, $\boldsymbol{M}_{\mathcal{H}}$ is invertible. There is a one-to-one map from $\{[r^{(j)}]_t\}_{j\in[n-t]}$ to $\{[u^{(i)}]_t\}_{P_i\in\mathcal{H}}$.

For each $j \in [n-t]$, $\mathcal{S}$ generates a random sharing $[r^{(j)}]_t$ based on the shares held by corrupted parties. For each honest party $P_i \notin \mathcal{H}$, $\mathcal{S}$ generates a random sharing $[u^{(i)}]_t$ based on the shares held by corrupted parties. Then, the sharings $\{[u^{(i)}]_t\}_{P_i\in\mathcal{H}}$ can be determined by $\{[r^{(j)}]_t\}_{j\in[n-t]}$ and $\{[u^{(i)}]_t\}_{P_i\notin\mathcal{H}}$. After $[u^{(i)}]_t$ for each honest party P_i is generated, $[v^{(i)}]_t, [\mu^{(i)}]_t, [\nu^{(i)}]_t$ are computed in the same way as that in $\mathbf{Hybrid_3}$.

Note that any misbehavior of corrupted parties can be detected and identified with overwhelming probability. In the case that each sharing $[u^{(i)}]_t$ dealt by a corrupted party P_i is consistent, by Lemma A.2, the resulting sharings $[r^{(1)}]_t, \ldots, [r^{(n-t)}]_t$ are random sharings. Therefore, the distribution is statistically close to $\mathbf{Hybrid_4}$. The negligible difference comes from the case where some misbehavior is not detected in Setup-Veri or identified in Setup-Single-Veri.

$\mathbf{Hybrid_6}$: Similarly to $\mathbf{Hybrid_5}$, in Output, $\mathcal{S}$ changes its strategy of generating $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$, $([v^{(i)}]_t, [\tilde{v}^{(i)}]_t)$ for each honest party P_i. In a nutshell, $\mathcal{S}$ first generates $\{([r^{(j)}], [\tilde{r}^{(j)}])\}_{j\in[n-t]}$ and then generates $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t), ([v^{(i)}]_t, [\tilde{v}^{(i)}]_t)$ for each honest party P_i. The distribution is statistically close to $\mathbf{Hybrid_5}$.

$\mathbf{Hybrid_7}$: In this hybrid, $\mathcal{S}$ further changes its strategy of generating $([r^{(j)}]_t, [\tilde{r}^{(j)}]_t)$ for all $j \in [n-t]$ in Output. In Refresh, $\mathcal{S}$ first samples a random sharing as $[s+r]_t$ based on the shares held by corrupted parties. Then, the sharing $[r]_t$ is set to be $[s+r]_t - [s]_t$. $\mathcal{S}$ samples a random sharing $[\tilde{r}]_t$ based on the secret $\tilde{r} = r$ and the shares held by corrupted parties. Note that this new way of generating $\{([r^{(j)}]_t, [\tilde{r}^{(j)}]_t)\}_{j\in[n-t]}$ does not change the distribution of $\{([r^{(j)}]_t, [\tilde{r}^{(j)}]_t)\}_{j\in[n-t]}$. Therefore the distribution remains the same as $\mathbf{Hybrid_6}$.

$\mathbf{Hybrid_8}$: In this hybrid, $\mathcal{S}$ first computes the sharings $\{([r^{(j)}]_t + \beta[\psi^{(j)}]_t, [\tilde{r}^{(j)}]_t + \beta[\tilde{\psi}^{(j)}]_t)\}_{j\in[n-t]}$ using $\{([u^{(i)}]_t + \beta[v^{(i)}]_t, [\tilde{u}^{(i)}]_t + \beta[\tilde{v}^{(i)}]_t)\}_{i\in[n]}$. In Refresh, $[z+\psi]_t$ can be computed by

$$[z+\psi]_t = \beta^{-1}(([s]_t + \beta[z]_t) + ([r]_t + \beta[\psi]_t) - ([s]_t + [r]_t)).$$

Note that $[s+r]_t$ and $[z+\psi]_t$ are prepared without using $[s]_t, [r]_t, [z]_t, [\psi]_t$. So far, Hand-off is fully simulated by $\mathcal{S}$. If $P_j \in \mathcal{C} \bigcup \mathcal{C}'$ is corrupted by $\mathcal{A}$, the view of P_j is prepared by $\mathcal{S}$ in the way we described above. Since $[\tilde{s}]_t$ may be used by RECONSTRUCTION, $\mathcal{S}$ prepares the whole sharing $[\tilde{s}]_t$. Recall that $[\tilde{s}]_t := (s+r) - [\tilde{r}]_t$ and $\tilde{s} = s$. $\mathcal{S}$ samples a random sharing $[\tilde{s}]_t$ based on the secret $\tilde{s} = s$ and the shares held by corrupted parties. The distribution is the same as $\mathbf{Hybrid_7}$.

$\mathbf{Hybrid_9}$: In this hybrid, $\mathcal{S}$ further changes its strategy of generating $[r^{(j)}]_t$ for all $j \in [n-t]$ in SETUP-OUTPUT. Recall that in $\mathbf{Hybrid_5}$, $\mathcal{S}$ first generates $\{[r^{(j)}]_t\}_{j\in[n-t]}$ and then generates $[u^{(i)}]_t$ for each honest party P_i.

In SETUP-FRESH, for an honest client, $\mathcal{S}$ first samples a random value as $s+r$. Then, $\mathcal{S}$ samples a random sharing $[s+r]_t$ based on the secret $s+r$ and the shares held by corrupted parties. The sharing $[r]_t$ is set to be $[s+r]_t - s$. Note that this new way of generating $\{[r^{(j)}]_t\}_{j\in[n-t]}$ does not change the distribution of $\{[r^{(j)}]_t\}_{j\in[n-t]}$. Therefore the distribution remains the same as $\mathbf{Hybrid_8}$.

$\mathbf{Hybrid_{10}}$: In this hybrid, $\mathcal{S}$ changes its strategy of generating z for an honest client. $\mathcal{S}$ chooses a random value as $z+\psi$ and then set $z = (z+\psi) - \psi$. Note that $s+r$ and $z+\psi$ are prepared without using s, z. So far, SETUP is fully simulated by $\mathcal{S}$. If $P_j \in \mathcal{C}$ is corrupted by $\mathcal{A}$, the view of P_j is prepared by $\mathcal{S}$ in the way we described above. Since $[s]_t$ may be used by RECONSTRUCTION, $\mathcal{S}$ prepares the whole sharing $[s]_t$. Note that if s is dealt by a corrupted client, the whole sharing $[s]_t$ has already been generated. For an honest client, recall that $[s]_t := (s+r) - [r]_t$. $\mathcal{S}$ samples a random sharing $[s]_t$ based on the secret s and the shares held by corrupted parties. The distribution is the same as $\mathbf{Hybrid_9}$.

$\mathbf{Hybrid_{11}}$: So far, the sharing $[s]_t$ is only used in RECONSTRUCTION and it is generated in the same way as that in RECONSTRUCTION. Therefore, $\mathcal{S}$ only generates the sharing $[s]_t$ when RECONSTRUCTION is invoked and Client is corrupted. Note that in this case $\mathcal{S}$ can learn the secret by querying $\mathsf{Ideal}_{\mathsf{safe}}$. The distribution is the same as $\mathbf{Hybrid_{10}}$.

Note that in the last hybrid, the simulator does not need to have access to the secrets to simulate the whole process. We conclude that the distribution of $\mathbf{Hybrid_{11}}$ is computationally indistinguishable to the distribution of $\mathbf{Hybrid_0}$.

E eWEB Security Proof

In the following, we prove that the eWEB construction (Figure 9) satisfies the security definition given in §3.2. Let ϵ_i denote the success probability of the adversary in hybrid $\mathbf{H_i}$.

For **secrecy**, we proceed with the proof using a number of hybrid games.

Proof. $\mathbf{H_0}$: This hybrid corresponds to the execution of the game using the HIDDENWITNESS construction as specified in Protocol 9.

$\mathbf{H_1}$: In this hybrid, SECRETSTORAGE and SECRETRELEASE functions are changed: instead of honestly constructing the CRS σ in step 2 of SECRETSTORAGE using $\mathsf{KeyGen}(1^k)$, the challenger executes the simulator $S_1(1^k)$ to get (σ', τ), and instead of honestly constructing a proof π in step 2 of SECRETRELEASE using $P(\sigma, pk, w)$, the challenger uses the simulator $S_2(\sigma, \tau, pk)$ to get a proof π'. The simulator $S = (S_1, S_2)$ is given by the unbounded computation zero-knowledge property of the used NIZK (as defined in §A.2). The challenger then uses σ' instead of σ and π' instead of π.

Lemma E.1. By the unbounded zero-knowledge property of the used NIZK system, there exists a negligible function $negl_1(\lambda)$ such that the following holds: $|\epsilon_1 - \epsilon_0| \leq negl_1(\lambda)$.

Proof. Given an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H_0}$ and $\mathbf{H_1}$, we define an adversary $\mathcal{B}$ on the unbounded zero-knowledge property of the used NIZK. $\mathcal{B}$ behaves like a challenger to $\mathcal{A}$, following the description of the security game defined in §3.2, except that instead of generating the crs σ on its own, $\mathcal{B}$ uses the one supplied by its own challenger. Then, whenever $\mathcal{A}$ requests $\mathcal{B}$ to create a secret release request for an honest identifier p_{id}, instead of providing the proof on its own, $\mathcal{B}$ chooses a witness $w \in W$, forwards the identifier p_{id} and the witness w to its challenger and supplies the resulting proof to $\mathcal{A}$. Upon $\mathcal{A}$ ending the game, $\mathcal{B}$ responds to its challenger with 0 (meaning the challenger was using the real prover algorithm) if $\mathcal{A}$'s response was H_0, and 1 otherwise. Note that if $\mathcal{B}$'s challenger was using the real prover algorithm, the game $\mathcal{A}$ is in is exactly $\mathbf{H_0}$, while if $\mathcal{B}$'s challenger was using the simulator, the game $\mathcal{A}$ is in is exactly $\mathbf{H_1}$. Thus, the advantage of the adversary $\mathcal{B}$ is exactly the same as the advantage of the adversary $\mathcal{A}$, and since the advantage of $\mathcal{B}$ is negligible by the unbounded zero-knowledge property of the used NIZK, the advantage of $\mathcal{A}$ is negligible as well. $\square$

$\mathbf{H_2}$: In this hybrid, SecretStorage is changed: instead of using the simulator $S_1(1^k)$, the challenger uses the simulator $SE_1(\sigma, \tau, pk)$, given by the simulation sound extractability property of the used NIZK (as defined in §A.2).

Lemma E.2. For the hybrids $\mathbf{H_1}$ and $\mathbf{H_2}$: holds: $|\epsilon_2 - \epsilon_1| = 0$.

Proof. Since the output of SE_1 is identical to S_1 when restricted to the first two parts (σ, τ) (as defined in §A.2), from the adversary's perspective nothing changed. $\square$

$\mathbf{H_3}$: In this hybrid, in addition to storing $(id, (F, \sigma))$ off-chain in step 3 of the SecretStorage procedure, the challenger stores this information internally. In step 5 of SecretStorage and steps 1 and 6 of SecretRelease, if private copy is available for the request, instead of checking the correctness of the hash, the off-chain data is compared directly with the private copy.

Lemma E.3. By the collision-resistance of the hash function used in our construction, there exists a negligible function $negl_3(\lambda)$ such that the following holds: $|\epsilon_3 - \epsilon_2| \leq negl_3(\lambda)$.

Proof. Given an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H_2}$ and $\mathbf{H_3}$, we define an adversary $\mathcal{B}$ on the collision-resistance of the used hash function. $\mathcal{B}$ behaves like a challenger to $\mathcal{A}$, as defined in hybrid $\mathbf{H_2}$, and additionally internally stores (F, σ) of the challenge request (note that this is the only storage request executed by the challenger acting as a user). Denote the internal copy by $(F_{intern}, \sigma_{intern})$. Then, when the challenger in the security game would get the offchain data (F', σ') and check the hash, $\mathcal{B}$ chooses a random bit b, and checks the hash of the offchain data if b is 0, and compares the offchain data to the internal one if b is 1. In any case, $\mathcal{B}$ sends (F', σ') and $(F_{inter}, \sigma_{inter})$ to its challenger.

If the offchain data is not altered, the check $\mathsf{requestHash} \overset{?}{=} H(F', \sigma')$ returns true, same as the check $(F_{intern}, \sigma_{inter}) \overset{?}{=} (F', \sigma')$. If the offchain data is altered and the hash of the altered data is not the same as of the original data, the results of the comparison are again the same - both return $false$. Thus, the only case where the challenger behaves differently in hybrids $\mathbf{H_2}$ and $\mathbf{H_3}$ (and the adversary thus has the chance of distinguishing $\mathbf{H_2}$ and $\mathbf{H_3}$) is the case where the offchain data is altered, but the hash of the altered data is still the same as that of the original. Thus, the advantage of the adversary $\mathcal{A}$ is at most the same as the winning probability of the adversary $\mathcal{B}$. Since by the collision-resistance property of the hash function used the winning probability of $\mathcal{B}$ is negligible, so is the advantage of $\mathcal{A}$. $\square$

$\mathbf{H_{4,(i,j)\in[n*d]\times[n*d]}}$: In this hybrid, all encrypted message sent from an honest party P_i to an honest party P_j are changed to encryptions of zero strings of the same length.

Lemma E.4. By the multi-message IND-CCA security of the encryption function used in our construction, there exists a negligible function $negl_{4,(i,j)}(\lambda)$ such that the following holds: $|\epsilon_{4,(i,j)} - \epsilon_{4,(i',j')}| \leq negl_{4,(i,j)}(\lambda)$, where $\epsilon_{4,(i',j')}$ denotes the advantage of the adversary in the hybrid previous to $\mathbf{H_{4,(i,j)}}$.

Proof. Given an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H_{4,(i,j)}}$ and the previous hybrid $\mathbf{H_{4,(i',j')}}$, we define an adversary $\mathcal{B}$ on the IND-CCA security of the used encryption scheme. $\mathcal{B}$ receives a public key pk from its challenger and sets the public key of P_j to be pk. Then, $\mathcal{B}$ behaves like a challenger to $\mathcal{A}$ as defined in the previous hybrid, except when P_i sends a message to P_j or when a message to P_j needs to be decrypted. In the former case, $\mathcal{B}$ submits the tuple $m, p_{id,i}$, where m is the corresponding message and $p_{id,i}$ is the identifier of P_i to its challenger and uses the resulting challenge instead of the encryption of the message. In the latter case, if the requested ciphertext is not the same as one of the challenge messages, $\mathcal{B}$ obtains the corresponding plaintext using its decryption oracle for pk. If the requested ciphertext c is one of the challenge messages and was sent by one of the adversarial parties, $\mathcal{B}$ responds with an error message "Authentication check failed". Note that this is possible since by the correctness property of the used encryption scheme we know that the message contained inside the ciphertext c is of the form $(m, p_{id,i})$ since c is a duplicate of a ciphertext produced by P_i. Thus, since c is coming from an adversarial party and the party identifiers are unique, we know that this ciphertext does not pass the authentication check as defined in Protocol 5. Thus, $\mathcal{B}$ acts the same way the challenger for $\mathcal{A}$ would act. Upon $\mathcal{A}$ ending the game, $\mathcal{B}$ responds to its challenger with 0 (meaning the challenge contained real messages) if $\mathcal{A}$'s response was $\mathbf{H_{4,(i',j')}}$, and 1 otherwise. Note that if $\mathcal{B}$'s challenger chose real messages, the game $\mathcal{A}$ is in is exactly $\mathbf{H_{4,(i',j')}}$, while if $\mathcal{B}$'s challenger chose zero strings, the game $\mathcal{A}$ is in is exactly $\mathbf{H_4, (i, j)}$. Thus, the advantage of the adversary $\mathcal{B}$ is exactly the same as the advantage of the adversary $\mathcal{A}$, and since the advantage of $\mathcal{B}$ is negligible by the IND-CCA security of the used encryption scheme, the advantage of $\mathcal{A}$ is negligible as well. ☐

$\mathbf{H_5}$: In this hybrid, the challenger switches from honestly executing the DPSS protocol to using the DPSS simulator S_{dpss} while honestly simulating $\mathbf{Ideal}_{\text{safe}}$ himself. In order to use S_{dpss}, the challenger simulates point-to-point channels of the DPSS protocol as follows:

- Whenever $\mathcal{A}$ sends a DPSS message to the challenger, the challenger decrypts the ciphertext using the secret key of a corresponding honest party to obtain a message of the form (m, p_{id}), and if it passes the authentication check (Protocol 5), eliminates the party identifier and forwards the resulting message m to S_{dpss}.

- Whenever the challenger receives a message m from S_{dpss} that is designated for an adversarial party p_r and is coming from an honest party p_s, the challenger sends the ciphertext $Enc_{pk_r}(m, p_{id,s})$ to $\mathcal{A}$, where $p_{id,s}$ is the identifier of the party p_s and pk_r is the public key of p_r.

- Whenever a DPSS message must be sent from one honest party to another, the challenger proceeds the same way as in $\mathbf{H_{4,(n*d,n*d)}}$ - sends encryption of the zero string of the same length as the corresponding message.

Additionally, upon receiving a secret storage request for the challenge from $\mathcal{A}$, the challenger asks the simulator to execute SECRETSTORAGE request as an honest client.

Finally, the challenger simulates $\mathbf{Ideal}_{\text{safe}}$ by storing a list L of identifier-secret pairs as follows:

- Once the challenger executed the secret storage request for the challenge, the challenger stores the pair (id, M_B) in L, where id is the identifier of the challenge request.
- Upon receiving a secret s from the simulator, the challenger stores the pair (id, s) in L, where id is the identifier of corresponding secret storage request.
- Upon S_{dpss} querying $\mathbf{Ideal}_{\text{safe}}$, the challenger looks up the list of the stored secrets and provides s if the pair (id, s) is in L.

Lemma E.5. By the secrecy of the used DPSS scheme, there exists a negligible function $negl_5(\lambda)$ such that the following holds: $|\epsilon_5 - \epsilon_{4,(n*d,n*d)}| \leq negl_5(\lambda)$.

Proof. Assume that there exists some adversary $\mathcal{A}$ able to distinguish between the hybrids $\mathbf{H_5}$ and $\mathbf{H_{4,(n*d,n*d)}}$. Then, we can construct an adversary $\mathcal{B}$ on the secrecy of the used DPSS scheme. $\mathcal{B}$ behaves as a challenger to $\mathcal{A}$. First, $\mathcal{B}$ sets up the public keys and the identifiers of the honest parties and submits these to $\mathcal{A}$. Then, whenever $\mathcal{A}$ chooses a new set of adversarial parties, $\mathcal{B}$ forwards its choice to its challenger. Whenever a DPSS message is sent by $\mathcal{A}$ to an honest party, $\mathcal{B}$ decrypts it using the secret key of this honest party to obtain a message of the form $(m, p_{id,s})$ and if it passes the authentication check, $\mathcal{B}$ forwards it to its challenger (otherwise $\mathcal{B}$ can safely abort outputting an error message "Authentication check failed"). Whenever $\mathcal{B}$ receives a message from its challenger that is designated for an adversarial party p_r (with a public key pk_r) and is coming from an honest party p_s, $\mathcal{B}$ sends the ciphertext $Enc_{pk_r}(m, p_{id,s})$ to $\mathcal{A}$, where $p_{id,s}$ is the identifier of the party p_s. Upon receiving a first secret storage request for the challenge from $\mathcal{A}$, $\mathcal{B}$ forwards M_b to its own challenger requesting to execute SetupPhase for M_b. Note that $\mathcal{B}$'s challenger does not provide $\mathcal{B}$ with messages that are directed to honest parties. $\mathcal{B}$ keeps track of the DPSS protocol and whenever an encryption of such message is expected by $\mathcal{A}$, $\mathcal{B}$ proceeds the same way as in $\mathbf{H_{4,(n*d,n*d)}}$ - sends encryption of the zero string of the same length. Note that if $\mathcal{B}$'s challenger uses the real DPSS protocol, the game $\mathcal{A}$ is in is exactly $\mathbf{H_{4,(n*d,n*d)}}$, while if $\mathcal{B}$'s challenger uses the simulator, the game $\mathcal{A}$ is in is exactly $\mathbf{H_5}$. Thus, the advantage of the adversary $\mathcal{B}$ is at least the same as the advantage of $\mathcal{A}$, and since the advantage of $\mathcal{B}$ is negligible by the secrecy of the DPSS scheme, the advantage of $\mathcal{A}$ is negligible (denote it by $negl_{5,1}(\lambda)$) as well. □

$\mathbf{H_6}$: In this hybrid, the challenger changes the way it is generating M_b and simulating $\mathbf{Ideal}_{\text{safe}}$: instead of saving (id, M_b) in L upon the SECRETSTORAGE request, the challenger generates and stores it only upon it is requested by the simulator.

Lemma E.6. For the hybrids $\mathbf{H_5}$ and $\mathbf{H_6}$ holds $|\epsilon_6 - \epsilon_5| = 0$.

Proof. Note that M_b is used only in the simulation of $\mathbf{Ideal}_{\text{safe}}$. The simulator of the DPSS scheme does not use a message stored in $\mathbf{Ideal}_{\text{safe}}$ until the reconstruction of this message is requested. Thus, it makes no difference whether the message is stored in L upon SECRETSTORAGE request or only once its reconstruction may be requested. □

Now, note that in the last hybrid the following holds:

Case 1: A provided a valid secret release request for function F for an identifier p_{id} of a dishonest user in step 8d) of the security game. In hybrid H_1 the challenger switched from providing real CRS and proofs to providing simulated CRS and proofs and in hybrid H_2 the challenger switched from using the simulator S_1 to using the simulator SE_1. Thus, in this case $\mathcal{A}$ interacts with the simulator (SE_1, S_2) and outputs a new (since the used party identifier is not honest) proof that is accepted by the challenger (acting as a verifier). By the proof of knowledge property of the used NIZK proof, there exists an extractor E_2 extracting a witness $w \leftarrow E_2(\sigma, \epsilon, pk, \pi)$. By the unbounded simulation sound extractability of the NIZK used the property $F(w) = true$ holds with the probability $1 - negl_{6,0}(\lambda)$ for some negligible function $negl_{6,0}(\lambda)$. Thus, in case 1, we extract a valid witness with an overwhelming probability $1-(negl_{6,0}(\lambda)+\mathsf{negl}(n)_5(\lambda)+\sum_{(i,j)\in[n*d]\times[n*d]}\mathsf{negl}(n)_{4,(i,j)}(\lambda)+\sum_{i\in[3]}\mathsf{negl}(n)_i(\lambda))$ and the security of the scheme holds in this case.

Case 2: A did not provide a valid secret release request for function F for a dishonest identifier p_{id} in step 8d) of the security game. In this case, we know that the adversary did not create a secret release request for the challenge in step 8d) of the security game. Thus, the DPSS simulator never requests M_b. Thus, M_b is never generated. Since we removed all information about the secret, the advantage of the adversary in winning in the last hybrid game is negligible (denote it by $negl_{6,1}(\lambda)$). Thus, the advantage of the adversary in H_0 is at most $\frac{1}{2} + negl_{6,1}(\lambda) + \mathsf{negl}(n)_5(\lambda) + \sum_{(i,j)\in[n*d]\times[n*d]}\mathsf{negl}(n)_{4,(i,j)}(\lambda) + \sum_{i\in[3]}\mathsf{negl}(n)_i(\lambda)$.

□

Now, we briefly outline why the scheme also has the **robustness** property assuming that the underlying off-chain storage is always available and it is not possible to change the data stored in the off-chain storage.

First, note that since hash function is a deterministic function and we assume that the data stored off-chain can not be altered, the test in line 5 of Protocol 6 always returns *true* for an honest client. If a client is dishonest and provided a wrong request hash, all honest miners abort the procedure and return $\perp$. In the first case, the miners subsequently store the information received through the DPSS protocol and by the robustness of the used DPSS scheme, after the execution of SecretStorage there exists a fixed secret and if the client that executed SecretStorage was honest, this secret is the same as the one chosen by the client.

Second, by the robustness property of the DPSS protocol, the fragmentation of sharing secret after the *SecretsHandoff* phase corresponds to the fragmentation of the secret after the *SecretStorage* phase.

Third, since hash function is a deterministic function and we assume that the data stored off-chain can not be altered, the tests in line 1, line 5 and line 6 of Protocol 8 always return *true* for honest clients that executed *SecretStorage* and *SecretRelease*. If the client executing *SecretRelease* is dishonest and provided a wrong request hash, all honest miners abort the procedure and return $\perp$. If the client who executed *SecretStorage* was dishonest, the honest client executing *SecretRelease* aborts the procedure and returns $\perp$. If both clients provided the correct request hashes and the client executing *SecretRelease* provided a valid proof, the client and the miners execute the DPSS **Reconstruction phase** and thus by the robustness of the DPSS protocol the client reconstructs the secret that existed at the end of the SecretStorage phase.

F Public Witness Protocol

Our second construction is the public witness protocol. In the public witness setting, the user requesting the release of the secret provides the witness for the secret release condition publicly, allowing everyone to reconstruct the secret.

In some applications it may be unnecessary to continue hiding the secret once a valid witness is submitted. Imagine a dead man's switch, where certain information must be publicly released once a person has been confirmed dead. Another example is a new episode of a top TV show that must be publicly released at a certain time. In this setting, we use a multisignature scheme [BGLS03] that allows a group of miners to jointly sign the data such that the verifying user is convinced that each member of the group participated in signing. The protocol can be altered as follows:

- Instead of using NIZK proofs, the witness can be submitted in clear.
- Instead of engaging in the DPSS **Reconstruction phase** with the user who provided a valid witness, miners post the corresponding reconstruction data off-chain and the hash of this data on-chain.
- Miners verify information posted by other miners and sign the verified shares using a multisignature scheme.
- Users retrieve the shares from blockchain and verify the signature. Using the retrieved information, users execute the DPSS **Reconstruction phase** to reconstruct the secret.

The complete construction is provided in Figure 31. As in §4.2, the construction uses Protocol 4 and Protocol 5 to handle point-to-point channels between the members of the system.

Protocol 28: SecretStorage-PublicWitness

1. The user computes hash of the decryption condition $\mathsf{requestHash} \leftarrow H(F)$, and publishes $\mathsf{requestHash}$ on blockchain. Let id be the storage identifier of the published request.
2. The user stores the tuple (id, F) offchain.
3. The user and the current members of the miner committee engage in DPSS **Setup Phase** protocol for the user's secret M.
4. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain, F from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of F:

$$\mathsf{requestHash} \overset{?}{=} H(F)$$

   If this is not the case, the committee member stops processing this request.

5. Otherwise, C_i stores $(id, \mathsf{dpss\text{-}data}_i)$ internally, where $\mathsf{dpss\text{-}data}_i$ is the DPSS data obtained after the DPSS **Setup Phase** procedure.

Protocol 29: SecretRelease-PublicWitness

1. The user computes hash $\mathsf{requestHash}^* \leftarrow H(id, w)$, and publishes $\mathsf{requestHash}^*$ on blockchain. Here, id is the storage identifier of the desired secret, and w is the witness to the decryption condition of this request on blockchain. Let id^* be the identifier of the published request.
2. The user stores the tuple $(id^*, (id, w))$ offchain.
3. Each committee member retrieves $\mathsf{requestHash}^*$ from the blockchain request with the identifier id^*, (id, w) from the offchain storage, and verifies that $\mathsf{requestHash}^*$ is indeed the hash of (id, w):
$$\mathsf{requestHash}^* \overset{?}{=} H(id, w)$$
If this is not the case, the committee member stops processing this request.
4. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain request with the identifier id, F from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of F:
$$\mathsf{requestHash} \overset{?}{=} H(F)$$
If this is not the case, the committee member stops processing this request.
5. Each committee member C_i retrieves the DPSS data $\mathsf{dpss\text{-}data}_i^{id}$ corresponding to the storage identifier id from its internal storage.
6. Each committee member C_i checks whether w is a valid witness for F:
$$F(w) \overset{?}{=} true$$
If so, C_i publishes $(p_{id,i}, id, \mathsf{dpss\text{-}data}_i^{id})$ on blockchain, where p_{id} is the party identifier of C_i.

Protocol 30: SecretReconstruction

1. For the secret storage identifier id, the user retrieves the corresponding tuples $(p_{id,i}, id, \mathsf{dpss\text{-}data}_i^{id})$ from blockchain, where $p_{id,i}$ is a party identifier of the miner C_i that was in the committee during the time of the corresponding secret release.
2. The user uses published shares $\mathsf{dpss\text{-}data}_i^{id}$ to reconstruct the secret as done in the DPSS **Reconstruction Phase** protocol.

Protocol 31: PUBLICWITNESS

1. A user can initiate secret storage by executing SECRETSTORAGE-PUBLICWITNESS (28).
2. A user can initiate secret release by executing SECRETRELEASE-PUBLICWITNESS (29).
3. After SECRETRELEASE-PUBLICWITNESS was successfully executed at least once, any user can obtain the secret using SECRETRECONSTRUCTION procedure (30).
4. In each round, miners execute SECRETSHANDOFF (7) procedure.

G Voting Application

In this section, we consider the "yes-no" voting scenario. Specifically, we consider the space of the secrets to be $\{-1, 1\}$, where "-1" stands for a "no" vote, and "1" stands for a "yes" vote. Note that, this requires the committee to be able to verify the correctness of the secrets shared by the clients. To this end, we will let each client first commit to its secret and then prove the correctness of the secret. In particular, we will show that the proof only contains one field element sending from the client to all parties in the committee.

G.1 Step 1: Preparing Commitments for the Secrets of Clients

Recall that in the setup phase, all parties in the committee first prepare a random sharing $[r]_t$. Then the client collects the whole sharing $[r]_t$, reconstructs the value r, and publishes $s+r$. Finally, all parties can compute $[s]_t = s + r - [r]_t$.

Let $[z]_t$ denote the random sharing used for the KZG commitment of $[s]_t$. Recall that g, h are two random generators of the group $\mathbb{G}$ in the public key PK of the KZG commitment scheme. Then $g^s h^z$ is the Pedersen commitment of s. A straightforward way of preparing this commitment is asking the client to compute it and publish the commitment to all parties. However, it will require the client to prove that it is the same value as the secret it shares to the committee.

We note that the Pedersen commitment is linear homomorphism. If all parties know the Pedersen commitment of r, after the client publishes $s + r$, all parties can locally compute the Pedersen commitment of s by themselves.

To this end, we modify the setup phase as following. In SETUP-DIST (Protocol 19), each party P_i will distribute two sharings $[u^{(i)}]_t, [v^{(i)}]_t$ and publish the commitment $\mathcal{C}om_{\text{KZG}}([u^{(i)}]_t; [v^{(i)}]_t)$. We require that each P_i further publishes the Pedersen commitment of $u^{(i)}$, i.e., $g^{u^{(i)}} h^{v^{(i)}}$. In SETUP-VERI (Protocol 21), all parties also compute the Pedersen commitment of $F(\lambda)$ and check its correctness. Similarly, in SETUP-SINGLE-VERI (Protocol 22), all parties compute the Pedersen commitment of $\mu^{(i)} + \lambda u^{(i)}$ and check its correctness. Finally, in SETUP-OUTPUT (Protocol 23), all parties locally compute the Pedersen commitments of $r^{(1)}, \ldots, r^{(n-t)}$.

We note that the Pedersen commitment is perfectly hiding. Therefore, the secrecy of the DPSS protocol still holds. In particular, for an honest party P_i, the simulator we constructed in Section D will generate $[u^{(i)}]_t + \beta [v^{(i)}]_t$ in clear, where β satisfies that $h = g^\beta$. Therefore, the simulator can compute the Pedersen commitment of $u^{(i)}$ by $g^{u^{(i)}+\beta v^{(i)}} = g^{u^{(i)}} h^{v^{(i)}}$.

G.2 Step 2: Proving the Correctness of the Secret

After Step 1, all parties will end up holding the Pedersen commitment of the secret s, i.e., $g^s h^z$. Now we want to verify that $s \in \{-1, 1\}$. This is equivalent to verify that $s^2 = 1$.

To this end, the client computes $w = g^{2sz} h^{z^2}$ and publishes w to all parties. Note that it can be done since the client knows s and z. Let β satisfy that $h = g^\beta$. To verify whether $s^2 = 1$, all parties check the following equation:

$$e(g^s h^z, g^s h^z) = e(g, g) \cdot e(h, w).$$

Correctness. To show correctness, note that

$$\begin{aligned} \text{LHS} &= e(g^{s+\beta z}, g^{s+\beta z}) = e(g, g)^{s^2+2\beta sz+\beta^2 z^2} \\ \text{RHS} &= e(g, g) \cdot e(g^\beta, g^{2sz+\beta z^2}) = e(g, g)^{1+2\beta sz+\beta^2 z^2}. \end{aligned}$$

Therefore, if the equation holds, then we have $s^2 = 1$.

Security. We first consider an corrupted client. Note that s, z are determined by the shares held by honest parties. In this setting, when receiving $(g, h = g^\beta)$ from the challenger, we show that if $s \notin \{-1, 1\}$ but the verification passes, then we can break the t-SDH assumption.

The simulator first prepares PK. This can be done by randomly samples α and computes $(g, g^\alpha, \ldots, g^{\alpha^t}, h, h^\alpha, \ldots, h^{\alpha^t})$. Then, the simulator simply follows the protocol using the real inputs. Recall that the simulator is able to extract s, z from the shares held by honest parties. Suppose that $w = g^x$ published by the corrupted client passes the verification. Then, we have

$$\begin{aligned} e(g^s h^z, g^s h^z) &= e(g, g)^{s^2+2\beta sz+\beta^2 z^2} \\ e(g, g) \cdot e(h, w) &= e(g, g) \cdot e(g^\beta, g^x) = e(g, g)^{1+\beta x}. \end{aligned}$$

Therefore, $x = \frac{s^2+2\beta sz+\beta^2 z^2-1}{\beta} = \frac{s^2-1}{\beta} + 2sz + \beta z^2$. The simulator computes

$$y = \left(\frac{w}{g^{2sz} \cdot h^{z^2}}\right)^{\frac{1}{s^2-1}} = g^{1/\beta}.$$

Finally, the simulator provides $\langle 0, y = g^{1/\beta} \rangle$ to the challenger.

We now consider an honest client. We show that the simulator we constructed in Section D is able to compute w. In this setting, the public key PK is prepared by the simulator. Recall that the simulator knows β and $s + \beta z$ when simulating Setup (Protocol 25). Therefore, the simulator can compute w by

$$g^{\frac{(s+\beta z)^2-1}{\beta}}.$$

Since the client is honest, s satisfies that $s^2 = 1$. Therefore, $\frac{(s+\beta z)^2-1}{\beta} = 2sz + \beta z^2$, which means that

$$g^{\frac{(s+\beta z)^2-1}{\beta}} = g^{2sz+\beta z^2} = g^{2sz} h^{z^2} = w.$$

Remark G.1. In [KZG10], there is another variant of the KZG commitment scheme which is computationally hiding. The Pedersen commitment $g^s h^z$ can be viewed as the KZG commitment of the polynomial $f(x) = s + z \cdot x$. Then, the client wants to prove that $f^2(0) = 1$. The above process can be seen as the client commits the polynomial $f^2(x)$ and opens the evaluation of $f^2(x)$ at point $x = 0$.

Since the verification of opening only requires $e(g^{f^2(\beta)}, g) = e(g^{f(\beta)}, g^{f(\beta)})$, the client does not need to publish the commitment $\mathcal{C}om = g^{f^2(\beta)}$.