Proactive Linear Integer Secret Sharing

Rune Thorbek

BRICS, Dept. of Computer Science, University of Aarhus

Abstract. In [3] Damgard and Thorbek proposed the linear integer secret sharing (LISS) scheme. In this note we show that the LISS scheme can be made proactive.

1 Introduction

Secret sharing schemes protect secrets by distributing them over different locations (parties). In a threshold-t scheme, the adversary is restricted to corrupt no more than t out of n parties throughout the lifetime of the scheme. If this assumption is fulfilled, then the adversary will not learn the secret. But in some long-lived schemes with sensitive secrets this guarantee may be insufficient.

In [9] Ostrovsky and Yung proposed the proactive model. In the proactive model the life span of the protocol is divided into separate time periods, where the adversary can corrupt at most t parties in each period. The set of corrupted parties may change from period to period, and the protocol must remain secure, even though every party may have been corrupt at some point. One way to achieve this, is by letting the parties re-randomize the shares they hold between each time period and erase the old shares. This procedure is called the *refreshment*. This should be done in such a way, that the parties are still guaranteed the correctness and the privacy of the scheme. Furthermore, to avoid gradual destruction of the secret by corruption of shares, it is necessary to be able to recover lost or corrupted shares without compromising the secrecy of any shares.

Proactive secret sharing has shown very useful in threshold RSA signature schemes [5], since it provides security in a long-lived system, e.g., for the master keys which cannot be periodically changed. While this area has been studied in great detail [4, 10, 8, 1, 6], we consider the LISS scheme in a proactive setting without a distributed exponentiation protocol in this chapter.

There are mainly two strategies to make a secret sharing scheme proactive. One way to implement the refreshment phase is to let every party re-share his share components and by the linearity of LISS reconstruct the secret in a new single secret sharing in LISS. Doing this straightforward has the disadvantage, that the share sizes grow very fast, since the LISS scheme is done over the integers and needs some additional bits in each round to "hide" the secret. Another strategy is to add a secret shared zero to the secret. In this strategy the share component sizes grow amortized less than one bit. While this solves the problem for a passive adversary, we need some means to detect tampered shares and recover them if we consider an active adversary. All this will be solved in this chapter.

2 Preliminaries

In a linear integer secret sharing [3] (LISS) scheme a dealer D can share a secret s from a publically know interval $[-2^l..2^l]$ over an access structure Γ between the party $\mathcal{P}$ such that only qualified subsets can reconstruct the secret while other subsets do not gain any information about the secret. More precisely,

Definition 1. *A LISS scheme is* correct*, if the secret can be reconstructed from shares of any qualified set in $A \in \Gamma$, by taking an integer linear combination of the shares with coefficient that depends only on the index set A.*

Definition 2. *A LISS scheme is* private*, if for any forbidden set $B \in \Delta$, any two secret $s, s' \in [-2^l..2^l]$, and independent random coins r and r', the statistical distance between the distributions of the shares $\{s_i(s, r, k) \mid i \in B\}$ and $\{s_i(s', r', k) \mid i \in B\}$ is negligible in the* security *parameter k.*

A *labeled matrix* consists of a $d \times e$ matrix M and a corresponding surjective function $\psi : \{1, \ldots, d\} \rightarrow \{1, \ldots, n\}$. We say that the i-th row is *labeled* by $\psi(i)$ or *owned* by party $P_{\psi(i)}$. For any subset $A \subset \mathcal{P}$, we let M_A denote the restriction of M to the rows labeled by some $P_{\psi(i)} \in A$. For any d-vector $\boldsymbol{x}$, we similarly denote $\boldsymbol{x}_A$ to be the restriction of entries i with $P_{\psi(i)} \in A$. For any two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, let $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ denote the inner product.

Definition 3 ([2]). *An* Integer Span Program (ISP) *for a monotone access structure Γ consists of a tuple $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$, where $M \in \mathbb{Z}^{d,e}$ is a labeled matrix with a surjective function $\psi : \{1, \ldots, d\} \rightarrow \{1, \ldots, n\}$, and the* target vector *$\boldsymbol{\varepsilon} = (1, 0, \ldots, 0)^T \in \mathbb{Z}^e$. Furthermore, for every $A \subseteq \mathcal{P}$ the following holds,*

- *for every* $A \in \Gamma$ *there exists a* reconstruction vector $\boldsymbol{\lambda} \in \mathbb{Z}^d$ *such that* $M_A^T \boldsymbol{\lambda} = \boldsymbol{\varepsilon}$.
- *for every* $A \notin \Gamma$ *there exists a* sweeping vector $\boldsymbol{\kappa} \in \mathbb{Z}^e$ *such that* $M_A \boldsymbol{\kappa} = \mathbf{0}$ *and* $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon} \rangle = 1$.

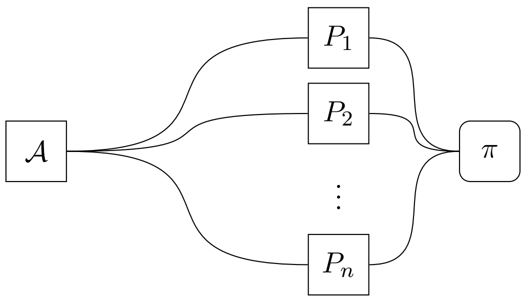
The size of $\mathcal{M}$ *is defined to be* d.

In [3] it was shown how to construct a correct and private LISS scheme from any ISP. For a given ISP we define $l_0 = l + \lceil \log_2(\kappa_{\max}(e-1)) \rceil + 1$, where $\kappa_{\max} = \max\{|a| \mid a \text{ is an entry in some sweeping vector }\}$. To share a secret $s \in [-2^l..2^l]$, we use a *distribution vector* $\boldsymbol{\rho}$ which is a uniformly random vector in $[-2^{l_0+k}..2^{l_0+k}]^e$ with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s$. The *share vector* is computed by $M\boldsymbol{\rho} = \boldsymbol{s} = (s_1, \ldots, s_d)^T$, where the *share component* s_i is given to party $P_{\psi(i)}$ for $1 \leq i \leq d$. The *share* of party P_j is the subset of share components $\boldsymbol{s}_{\{P_j\}}$.

3 Proactive Security

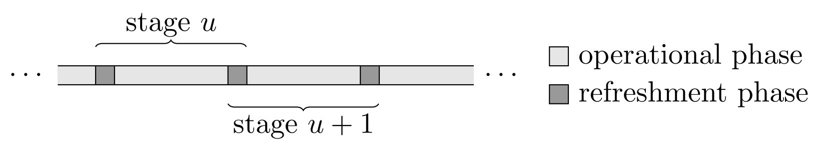
All the entities are Interactive Turing machines (ITM). The parties $P_1, \ldots, P_n$ are PPT ITMs which proactively secret share a secret. The adversary $\mathcal{A}$ attacking the protocol is a ITM with unbounded computing power. See Section 5.1 for further discussion on the computing power of the adversary.

For simplicity we assume authenticated secure point-to-point channels between all parties. This assumption will be discussed in more detail in section 6. Furthermore, we assume a synchronous network, which proceeds in *rounds*.



Since we assume point-to-point channels, the adversary cannot see the communication between the parties. The adversary only sees the view of the parties she corrupts, i.e., the messages received by the parties.

A *proactive* secret sharing scheme is divided into phases. We distinguish between an *operational* phase and a *refreshment* phase, which occur alternately. The operational phases are where the secret sharing can be used for whatever purpose it was intended, while the refreshment phases are used to re-randomize the secret sharing, such that attacks in different phases will not be able to benefit from each other. A *stage* consists of an *opening* refreshment phase, an operational phase, and a *closing* refreshment phase. Note, that if a refreshment phase is the closing refreshment phase of stage u, then it is the opening refreshment phase of stage $u + 1$.



The adversary $\mathcal{A}$ decides when the refreshment phases start by sending a command to all parties. While the refreshment ends when all honest parties have output a special symbol indicating the end of refreshment.

In a proactive secret sharing scheme realizing Δ the adversary may adaptively corrupt any party during any phase, with the restriction that the set C of corrupted parities is in Δ in every stage. Hence, if a party is corrupted during a refreshment phase, she is considered corrupt in both stages that the phase belongs to.

If a party is corrupted during an operational phase the adversary is given the view of the party from the beginning of the phase. Furthermore, if a party is corrupted during a refreshment phase, then the adversary is given the view starting from the beginning of the preceding operational phase.

The adversary can *decorrupt* a party at the end of any operational phase, and the party will be considered honest again. We distinguish between a passive and an active adversary. When a *passive* adversary corrupts a party, she is given the internal state of the party and the control. However the passive adversary follows the protocol and on decorruption she leaves the internal state of the party in a consistent state with the protocol. When an *active* adversary corrupts a party, she is given the internal state of the party, but she may deviate arbitrary from the protocol and leave the party on decorruption in an arbitrary internal state.

The parties are assumed to get randomness from an incorruptible source and are seeded with new fresh randomness whenever required. I.e. after decorruption the adversary does not know the random choices made by the party.

We need the following definition in order to formally define proactive security.

Definition 4. *Let $\boldsymbol{y}$ be a share vector and let H be the set of all honest parties. If $\langle \boldsymbol{y}, \boldsymbol{\lambda}_A \rangle = y$ for all qualified $A \subseteq H$, where $\boldsymbol{\lambda}_A$ is the reconstruction vector for A and y is a fixed value, then $\boldsymbol{y}$ is said to have* correct reconstructability*.*

Definition 5. *A proactive secret sharing scheme is* robust*, if it has correct reconstructibility for the same value in each operational phase.*

Definition 6. *A proactive secret sharing scheme realizing Δ is* proactively private*, if the adversary may corrupt and decorrupt, as described above, is private in all phases.*

4 Proactive Protocol

The protocol is as follows.

Protocol Refresh

Initially a dealer has computed sharing vector $(s_1, \ldots, s_d)^T = M\boldsymbol{\rho}$, where $\boldsymbol{\rho} \in [-2^{l_0+k}..2^{l_0+k}]^e$ was chosen at random with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s$ the secret.

1. For $i = 1, \ldots, n$ party P_i secret shares a zero, using uniformly random $\boldsymbol{\rho}_0 \in [-2^{l_0+k}..2^{l_0+k}]^e$, with the restriction that $\langle \boldsymbol{\rho}_0, \boldsymbol{\varepsilon} \rangle = 0$, to obtain share vector by $(s_{i_1}, \ldots, s_{i_d})^T = M\boldsymbol{\rho}_0$ and for $j = 1, \ldots, d$ send s_{i_j} to party $P_{\psi(j)}$.
2. For each $i = 1, \ldots, d$ party $P_{\psi(i)}$ lets $s_i^{(\text{new})} = s_i + \sum_{j=1}^{n} s_{j_i}$ be the new share component.

Note 1. If the above protocol is used r times, then each share component size grows at most $\log(n) + \log(r) + 1$ bits, i.e., amortized each share component grows $O(\log(r)/r)$ bits.

Theorem 1. *If the* Refresh *protocol is used polynomial many times, then it is robust and proactively private for a passive adversary.*

Proof. By inspection of the protocol it follows that the protocol is robust.

In order to prove that the protocol provides proactive privacy we provide a simulator. In each phase the adversary can corrupt and decorrupt as many parties she wants with the restrictions described in Section 3.

The idea behind the simulator is as follows. The simulator generates a LISS secret sharing of 0 and keeps the shares on behalf of each honest party and provides the adversary with the shares for the corrupted parties. The simulator then interacts in the protocols on behalf of each honest party throughout the run. When a party is corrupted, the simulator will have the required data of the honest party internal state and provides the adversary with it. On decorruption of a party, the simulator will receive the internal state from the adversary, and hence, has a consistent internal state of the now honest party.

For notational convenience we let $0, 1, \ldots, u$ denote the phases, and let $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_u$ denote the internal states of the honest parties. Note, that the internal state of honest party P_i in phase j is given by $(\boldsymbol{s}_j)_{\{P_i\}}$. Furthermore, we let the entries in $\boldsymbol{s}_j$ owned by the corrupted parties all contain the last known state. If j is a refreshment phase, then we let $\boldsymbol{s}_j$ denote the final state of the phase, i.e., $\boldsymbol{s}_j = \boldsymbol{s}_{j+1}$. Finally, let Δ denote the adversary structure for the LISS scheme.

The simulator acts as follows when ever required.

- In the first phase the simulator chooses $\boldsymbol{\rho}_0 \in [-2^{l_0+k}..2^{l_0+k}]^e$ uniformly at random with the restriction that $\langle \boldsymbol{\rho}_0, \boldsymbol{\varepsilon} \rangle = 0$. Computes $\boldsymbol{s}_0 = M\boldsymbol{\rho}_0$, the internal states of all parties. If the parties $A \in \Delta$ are corrupted at this point, the adversary is provided with $(\boldsymbol{s}_0)_A$.
- At the beginning of each refreshment phase the simulator runs the Refresh protocol on behalf of each honest party.
- When a party P_i is corrupted in phase j, the simulator provides the adversary with $(\boldsymbol{s}_j)_{\{P_i\}}$.
- When a party P_i is decorrupted in phase j the simulator updates the state $(\boldsymbol{s}_j)_{\{P_i\}}$ with the provided data from the adversary.
- In the end of the last phase u when the secret s is revealed, the simulator lets $\boldsymbol{s}'_u = \boldsymbol{s}_u + sM\boldsymbol{\kappa}$, where $\boldsymbol{\kappa}$ is the sweeping vector for A, the set of currently corrupted parties, and reveals $(\boldsymbol{s}'_u)_{A^\complement}$ to the adversary.

We need to show that this view is statistically close to a run of the real protocol.

First two observations of the corruption in the phases. First, since only corruptions and decorruptions happens in the operational phases, i.e., the internal state does not change of any party, we can assume that all parties are corrupted in the beginning of the phase. Secondly, since we assume that all corruptions happen in the beginning of each round and the Refresh protocol takes one round, we can also assume that all corruptions of the refreshment phase happen in the beginning of the phase.

Let $A_0, A_1, \ldots, A_u$ denote the unqualified set of corrupted parties in each phase. Let $\boldsymbol{\kappa}_0, \boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_u$ be the sweeping vectors for $A_0, A_1, \ldots, A_u$, respectively.

Assume first that the adversary only has corrupted parties in the operational phases, that is, in each refreshment phases all parties are honest. First of all, each operational phase isolated is private. We need to show, that given two views of operational phases, the scheme is still private.

Let share vector $\boldsymbol{s}'_i = \boldsymbol{s}_i - sM\boldsymbol{\kappa}_i$. Then note, that the adversary knows $(\boldsymbol{s}_i)_{A_i} = (\boldsymbol{s}'_i)_{A_i}$ for all $i = 0, 1, \dots, u$. However the share vectors are also dependent,

$$\begin{aligned}
\boldsymbol{s}_1 &= \boldsymbol{s}_0 + \sum_{i=1}^{n} \boldsymbol{z}_i^{(0)} \\
\boldsymbol{s}_2 &= \boldsymbol{s}_1 + \sum_{i=1}^{n} \boldsymbol{z}_i^{(1)} \\
&\vdots \\
\boldsymbol{s}_u &= \boldsymbol{s}_{u-1} + \sum_{i=1}^{n} \boldsymbol{z}_i^{(u-1)},
\end{aligned}$$

where $\boldsymbol{z}_i^{(j)}$ is a zero-sharing for all $i = 1, \dots, n$ and $j = 0, \dots, u-1$.

Consider the following.

$$\begin{aligned}
\boldsymbol{s}'_0 &= \boldsymbol{s}_0 + sM\boldsymbol{\kappa}_0 \\
\boldsymbol{s}'_1 &= \boldsymbol{s}'_0 - sM\boldsymbol{\kappa}_0 + \sum_{i=1}^{n} \boldsymbol{z}_i^{(0)} + sM\boldsymbol{\kappa}_1 \\
\boldsymbol{s}'_2 &= \boldsymbol{s}'_1 - sM\boldsymbol{\kappa}_1 + \sum_{i=1}^{n} \boldsymbol{z}_i^{(1)} + sM\boldsymbol{\kappa}_2 \\
&\vdots \\
\boldsymbol{s}'_u &= \boldsymbol{s}'_{t-1} - sM\boldsymbol{\kappa}_{u-1} + \sum_{i=1}^{n} \boldsymbol{z}_i^{(u-1)} + sM\boldsymbol{\kappa}_u,
\end{aligned}$$

where the view of the adversary is exactly the same, but the secret s is proactively shared instead of 0. Consider,

$$\boldsymbol{s}'_j = \boldsymbol{s}'_{j-1} - sM\boldsymbol{\kappa}_{j-1} + \sum_{i=1}^{n} \boldsymbol{z}_i^{(j-1)} + sM\boldsymbol{\kappa}_j,$$

which also can be represented by the distribution vectors,

$$\boldsymbol{\rho}'_j = \boldsymbol{\rho}'_{j-1} - s\boldsymbol{\kappa}_{j-1} + \sum_{i=1}^{n} \boldsymbol{\rho}_{z,i}^{(j-1)} + s\boldsymbol{\kappa}_j,$$

such that $\boldsymbol{s}'_j = M\boldsymbol{\rho}'_j, \boldsymbol{s}'_{j-1} = M\boldsymbol{\rho}'_{j-1}$ and $\boldsymbol{z}_i^{(j-1)} = M\boldsymbol{\rho}_{z,i}^{j-1}$ for $i = 1, \ldots n$.

Note that the term $s\boldsymbol{\kappa}_j$ does not influence the view of the adversary, and the term $s\boldsymbol{\kappa}_{j-1}$ can be subtracted from an honest 0-sharing $\boldsymbol{z}_i^{(j-1)}$ and therefore not change the view of the adversary. Furthermore note, that the adversary does not have any partial information on any of the $\boldsymbol{z}_i^{(j-1)}$, since we assumed that the adversary only corrupted parties in the operational phase.

That is, if the protocol is only used polynomial many times then the privacy of LISS ensures that the views are indistinguishable with all but negligible probability in the security parameter k if the adversary has no corrupted parties in the refreshment phases.

We only need to argue, that if the adversary corrupts a party during a refreshing phase, it does not help her. However consider the following.

$$\boldsymbol{\rho}'_j = \boldsymbol{\rho}'_{j-1} - s\boldsymbol{\kappa}_{j-1} + \sum_{i=1}^{n} \boldsymbol{\rho}_{z,i}^{(j-1)} + s\boldsymbol{\kappa}_j.$$

If j is a refreshment phase, then note, that if parties $A_j \subset \mathcal{P}$ were corrupted during that phase, then $A_j = A_{j-1} \cap A_j \cap A_{j+1}$. Hence, the two terms $s\boldsymbol{\kappa}_{j-1}$ and $s\boldsymbol{\kappa}_j$ do not change any of the corrupted parties shares. We therefore conclude, that the view is exactly the same and therefore the adversary does not profit from the corruption of the parties in A during the refreshment phase.

5 Active Security

5.1 Verifiable Secret Sharing

In order to make the LISS scheme and the Refresh protocol secure against an active adversary we need a linear integer commitment scheme. Let $C_a = \mathsf{com}(a)$ denote the commitment of a. Strictly speaking a commitment $C_a = \mathsf{com}(a, r)$ includes the commitment a and some randomness r. One can *open* a commitment by revealing a and r, then everybody can check that $\mathsf{com}(a, r) = C_a$. For simplicity we exclude the randomness used in the commitment.

A commitment $\mathsf{com}(a) = C_a$ is *binding* if the committer cannot open the commitment to any other value $b \neq a$. A commitment $\mathsf{com}(a) = C_a$ is *hiding* if C_a does not reveal anything about a. Both the binding and the hiding property come in a *computational*, *statistical*, and *perfect* flavor. Since we do not consider any specific commitment scheme one should just remember, that the security is no stronger than the commitment scheme.

By linearity we mean, given two commitments $C_a = \mathsf{com}(a)$ and $C_b = \mathsf{com}(b)$ and constant c it is possible to compute commitment $C = \mathsf{com}(a + cb)$, such that being able to open $\mathsf{com}(a)$ and $\mathsf{com}(b)$ makes it possible to open $\mathsf{com}(a + cb)$. We use the following notation for simplicity $C = C_a C_b^c$.

Given a linear integer commitment scheme com, then verifiable secret sharing (VSS) for the LISS scheme is constructed as follows. A distribution vector is chosen $\boldsymbol{\rho} \in [-2^{l_0+k}..2^{l_0+k}]$ with $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s \in [-2^l..2^l]$ the secret to be VSSed. Then commit to $C_1 = \mathsf{com}(s), C_2 = \mathsf{com}(\rho_2), \ldots, C_e = \mathsf{com}(\rho_e)$ and broadcast them to all parties involved. Then compute $\boldsymbol{s} = M\boldsymbol{\rho}$ for ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$. For $i = 1, \ldots, n$ send share $\boldsymbol{s}_{\{P_i\}}$ to P_i.

By the linearity of the scheme every party can check that the received share components are consistent with the broadcast commitments. Simply note,

$$\mathsf{com}(s_i) = C_1^{m_{i,1}} C_2^{m_{i,2}} \cdots C_e^{m_{i,e}},$$

where $M = [m_{i,j}]$. Furthermore, this ensures the parties that the VSS value is consistently opened.

5.2 The Refresh Protocol with Active Security

The simple observation is, that in order for a party to convince the others that he secret shares a 0, he simply uses the trivial commitment 1 which commits to 0 using randomness 0.

In the following protocol we first assume, that all honest parties internal states are consistent, that is, their shares and commitments are correct. We will come back to this issue in the next subsection.

Protocol $\mathsf{Refresh}$

Initially a dealer has computed sharing vector $(s_1, \ldots, s_d)^T = M\boldsymbol{\rho}$, where $\boldsymbol{\rho} \in [-2^{l_0+k}..2^{l_0+k}]^e$ was chosen at random with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s$ the secret. Furthermore, all parties have commitments $C_1 = \mathsf{com}(s), C_2 = \mathsf{com}(\rho_2), \ldots, C_e = \mathsf{com}(\rho_e)$.

1. *Refreshment.* For $i = 1, \ldots, n$ party P_i secret shares a zero, using uniformly random $\boldsymbol{\rho} \in [-2^{l_0+k}..2^{l_0+k}]^e$, with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = 0$, to obtain share vector by $(s_{i_1}, \ldots, s_{i_d})^T = M\boldsymbol{\rho}$. He broadcasts commitments $C_{i_2} = \mathsf{com}(\rho_2), \ldots, C_{i_e} = \mathsf{com}(\rho_e)$ and for $j = 1, \ldots, d$ send s_{i_j} to party $P_{\psi(j)}$.
2. *Verification.* For $j = 1, \ldots, n$ and $i = 1, \ldots, d$ party $P_{\psi(i)}$ verifies that $\mathsf{com}(s_{j_i}) = C_{j_2}^{m_{j,2}} \cdots C_{j_e}^{m_{j,e}}$, if not, then broadcast $(\mathsf{Complaint}, P_j)$
3. *Accusation.* If P_j receives $(\mathsf{Complaint}, P_j)$ from P_i, then P_j broadcasts all private information he sent to P_i. If P_j sends information

that fails the verification in step 2 or sends nothing, then P_j is added to an initially empty set B, otherwise P_i is added to B. If B is qualified, then abort the protocol.

4. *Update.* For each $i = 1, \dots, d$ party $P_{\psi(i)}$ lets $s_i^{(\text{new})} = s_i + \sum_{j=1, j \notin B}^{n} s_{j_i}$ be the new share component, and $C_j^{(\text{new})} = C_j \prod_{i=1, j \notin B}^{n} C_{j_i}$ for $j = 2, \dots, e$.

For now, we just assume that information broadcast in step 3 is the same send privately to P_i. See Note 3 and Section 6 for a solution of the problem. Then finally note, if the access structure is $Q3$, then the problem can be solved by adding both parties to P_i and P_j to B in step 3.

If a corrupt party P_i complains about an honestly generated zero-sharing, then obviously she will be added to B. If a corrupt party P_j shares an inconsistent zero-sharing, then an honest party will complain, and P_j will be added to B. Hence, B *cannot* become qualified.

While this ensures that the refreshment is done correctly, this does *not* ensure that the internal states are correct of the honest parties.

5.3 Detection of Corrupted Shares and Recovery of Shares

The active adversary can leave the internal state of the decorrupted party in an arbitrary way. First of all, a party does not know whether he was corrupted or not, hence we need a detection protocol, which can clarify this matter for each currently honest party.

Consider the following protocol.

Protocol Detect

Each party P_i internal state consists of a share $\boldsymbol{s}_{\{P_i\}}$ and commitments $(S^{(i)}, R_2^{(i)}, \dots, R_e^{(i)})$ of supposedly $\boldsymbol{\rho} = (s, \rho_2, \dots, \rho_e)^T$ for $\boldsymbol{s} = M\boldsymbol{\rho}$ and a Recover bit which is set if he knows the internal state has been tampered.

1. Each party P_i broadcasts commitments $(S^{(i)}, R_2^{(i)}, \dots, R_e^{(i)})$
2. Each party P_i collects

$$B = \{P_j \in \mathcal{P} \mid (S^{(j)}, R_2^{(j)}, \dots, R_e^{(j)}) = (S^{(i)}, R_2^{(i)}, \dots, R_e^{(i)})\}.$$

If B is not qualified or the share components known by P_i are not consistent with $(S^{(i)}, R_2^{(i)}, \dots, R_e^{(i)})$, then the state of P_i is corrupted and flips on the Recover bit.

Lemma 1. *Let Δ be a $Q2$ adversary structure, let C be the set of corrupted parties, and D be the set of decorrupted parties. If the adversary cannot break the binding property of the commitment scheme in use and $C \cup D \in \Delta$, then the* Detect *protocol will put all honest parties with tampered internal state in* Recover *state.*

Proof. Divide the parties in three groups. Let H be the set of parties which has never been corrupted, C the set of parties which are corrupted, and D the set of parties which are honest but are decorrupted. Note that the sets are mutually disjoint but the union cover the entire set of parties.

The set H will follow the protocol and their set collected in step 2 of the protocol will be qualified since the adversary structure Δ is $Q2$, i.e., $H = (C \cup D)^{\mathsf{C}}$ is qualified.

The set D do not necessarily have tampered input. But if a party has tampered commitments he cannot receive a qualified set of the same commitments, since they must come from other parties in the set C or D. If only the share components are tampered, they will not open the right combinations of the commitments under the assumption that the adversary cannot break the binding property of the commitment scheme.

The recovery process can be achieved by the following protocol. The protocol assumes that the honest parties agree on the commitments. This can be achieved by running the Detect protocol.

Protocol Recovery

The honest parties mutually agree on the following commitments, $C_1 = \mathsf{com}(s), C_2 = \mathsf{com}(\rho_2), \ldots, C_e = \mathsf{com}(\rho_e)$, where $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T \in [-2^{l_0+k}..2^{l_0+k}]^e$ was chosen at random with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s \in [-2^l..2^l]$. Then for each $i = 1, \ldots, n$ P_i received $\boldsymbol{s}_{\{P_i\}}$ privately from the dealer, where $\boldsymbol{s} = (s_1, \ldots, s_d)^T = M\boldsymbol{\rho}$ for an ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$, where $M \in \mathbb{Z}^{d,e}$. Let $l^{(\max)}$ be the maximal bit-length of any share component.

1. *Recovery.* For each $i = 1, \ldots, d$ party $P_{\psi(i)}$ VSS share component s_i by choosing distribution vector $\boldsymbol{\rho} = (s_i, \rho_2, \ldots, \rho_e)^T \in [-2^{l_0^{(\max)}+k}..2^{l_0^{(\max)}+k}]^e$ uniformly at random, with $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s_i$. Computes share vector $\boldsymbol{s}_i = M\boldsymbol{\rho}$. Broadcasts commitments $C_2^{(i)} = \mathsf{com}(\rho_2), \ldots, C_e^{(i)} = \mathsf{com}(\rho_e)$. For $j = 1, \ldots, n$ send $(s_i)_{\{P_j\}}$ to party P_j.
2. *Verification.* When any party P_j receives broadcast commitments $(C_2^{(i)}, \ldots, C_e^{(i)})$ from $P_{\psi(i)}$ along with privately received shares

$(\boldsymbol{s}_i)_{\{P_j\}}$ verify for each share components $s_\iota \in (\boldsymbol{s}_i)_{\{P_j\}}$ that $\mathsf{com}(s_\iota) = C_{s_i}^{m_{\iota,1}}(C_2^{(i)})^{m_{\iota,2}} \cdots (C_e^{(i)})^{m_{\iota,e}}$, where $C_{s_i} = C_1^{m_{i,1}} C_2^{m_{i,2}} \cdots C_e^{m_{i,e}}$ and $M = [m_{i,j}]$. If not, broadcast $(\mathsf{Complaint}, P_i)$.

3. *Accusation.* When party P_i receives broadcast $(\mathsf{Complaint}, P_i)$ from P_j he broadcasts all the private information send to P_j in the recovery step (step 1).
4. *Verification of Accusation.* When a party receives a opened share of P_j from party P_i in step 3 he verifies if the opened share can open the correct commitments, if not, P_i is added to an initially empty list B otherwise P_j is added to list.
5. *Reconstruction.* They chose a reconstruction vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_d)^T$ which excludes all parties in B and for all $i = 1, \ldots, d$ party $P_{\psi(i)}$ lets $s_i^{(\text{new})} = \sum_j \lambda_j s_{j_i}$ be the new share. Let the commitments $C_1^{(\text{new})} = \prod_{i=1}^d C_{s_i}^{\lambda_i}$ and for $j = 2, \ldots, e$ let $C_j^{(\text{new})} = \prod_{i=1}^d (C_j^{(i)})^{\lambda_i}$. Finally, clear the internal state for everything except the new commitments and share.

Note 2. If the access structure involved in above protocol is $Q3$, then step 3 and 4 in the protocol are not necessary, since we can just add both parties in the set B.

Note 3. If step 3 and 4 are included in the protocol, then we need a way to ensure, that the information broadcast in step 3 is the same as party P_j received in step 1. For simplicity we assume that this is forfilled. For further discussion and solutions of the problem, see Section 6.

In the refreshment phase the parties should run the Detect and the Refresh protocol in parallel, that way each honest party knows whether his internal state has been tampered.

Note, that the Recover protocol assumes that the internal states of the commitment are consistent, this can be achieved by running the Detect protocol in parallel. Furthermore, an honest party with tampered shares will be included in the list B from step 4 in the Recover protocol, since at the point of verification every honest party agrees on which commitments are the correct ones.

Remark 1. The reconstruction vector chosen in step 5 can be chosen deterministically based on the set B from step 4, hence, no further communication is required.

Theorem 2. *If the commitment scheme is binding then all honest parties will have consistent internal state after a parallel run of the* Detect *and the* Refresh *protocol.*

Proof. First we consider an honest party with consistent internal state, i.e., a state which the adversary has not tampered. If he receives some counterfeit shares from a party, then he will complain and under the assumption in Note 3 the accused party will broadcast the same information such that all parties can verify that the claim is correct. Note, that the problem is only relevant if the access structure is not $Q3$.

An honest party with tampered internal state will either discover the inconsistency due to the Detect protocol or otherwise he will be excluded from the protocol by complaints in step 2. Finally note, that he will receive correctly generated shares from all honest parties and recover the state.

6 Dynamically Authenticated Secure Point-to-Point Channels

The assumption of authenticated secure point-to-point channels between all parties might not be realistic even in the case of a passive adversary. In real life this model would be implemented by using some kind of public key infrastructure. While the adversary breaks into the different parties, she could steel the private key. After a couple of stages she would know all the parties private keys, and hence, be able to decrypt all messages send in the network.

If we consider an active adversary, then two further problems appear. Firstly, how to resolve the problem if the adversary removes or changes the private key and/or uses it. Secondly, the problem pointed out in Note 3.

Herzberg et al. [7] proposed to have initialized public/secret key pairs, such that party P_i knows all parties correct public keys $pk_1^{(0)}, \ldots, pk_n^{(0)}$ and has his own private key $sk_i^{(0)}$. Then at each refreshment phase, each party P_i generates a new public/secret key pair $sk_i^{(1)}/pk_i^{(1)}$ and signs the new public key by $sk_i^{(0)}$ and broadcasts it.

If the adversary knows $sk_i^{(u)}$ but has decorrupted party P_i, i.e., party P_i is honest again, then obviously she can decrypt all messages encrypted to P_i. But when P_i decides to generate a new pair $pk_i^{(u+1)}$ and $sk_i^{(u+1)}$ and broadcasts $pk_i^{(u+1)}$ signed with $pk_i^{(u)}$, then the adversary does not know $sk_i^{(u+1)}$ since the adversary does not know the randomness used to generate the key pair. This solves the problem with passive corruption.

With an active adversary we first consider the problem pointed out in Note 3. It can be solved by using the public key infrastructure described above. Simply, instead of party P_i sends private messages to, say, P_j, then P_i encrypts the private messages under public key $pk_j^{(u)}$, signs everything he sends under his secret key $sk_i^{(u)}$, and broadcasts everything. Then when P_i is asked to open the message send to P_j, he simply broadcasts the messages and the randomness used to encrypt under the public key $pk_j^{(u)}$ of P_j. If P_i fails to do this, he is considered corrupt, otherwise the accuser P_j is considered corrupt. Note again, that this is only a problem when we consider access structures which are not $Q3$.

Consider the problem when the adversary $\mathcal{A}$ uses some decorrupted honest party's P_i secret key $sk_i^{(u)}$, while the secret key has not been tampered in the internal state by the adversary, i.e., P_i still has the same copy of $sk_i^{(u)}$. Herzberg et al. [7] suggest to let the honest party P_i send a signed disqualification of his own secret key. Then a reboot procedure should be started, where all parties are rebooted, i.e., the adversary is removed from all of them, and a new consistent public key infrastructure is provided to the parties. The reason this strategy is necessary is that the parties cannot distinguish which messages come from the adversary $\mathcal{A}$ and which come from the honest party P_i.

Finally consider the case where the adversary $\mathcal{A}$ tampers or removes the private key $sk_i^{(u)}$ of P_i. This obviously makes it impossible for P_i to sign a message to start a reboot process. The simple solution is to send an unsigned reboot symbol to initiate the reboot. This opens up for the possibility that the adversary can start a reboot process without corrupting any party. But to our knowledge there is no way to elegantly handle this problem.

7 Acknowledgments

Ivan Damgård, Martin Geisler, Mikkel Krøigård, and Gert Læssøe Mikkelsen.

References

1. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 1999.
2. R. Cramer and S. Fehr. Optimal black-box secret sharing over arbitrary abelian groups. In M. Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 2002.
3. I. Damgård and R. Thorbek. Linear integer secret sharing and distributed exponentiation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2006.
4. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *FOCS*, pages 384–393, 1997.

5. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive rsa. In B. S. K. Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 440–454. Springer, 1997.
6. Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptive security for the additive-sharing based proactive rsa. In K. Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 240–263. Springer, 2001.
7. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 1995.
8. S. Jarecki and N. Saxena. Further simplifications in proactive rsa signatures. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 510–528. Springer, 2005.
9. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *PODC*, pages 51–59, 1991.
10. T. Rabin. A simplified approach to threshold and proactive rsa. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 1998.