Asynchronous Error-Decodable Secret-Sharing and Its Application

Ashish Choudhury(✉)

International Institute of Information Technology, Bangalore, India
ashish.choudhury@iiitb.ac.in

Abstract. In this paper, we study error-decodable secret-sharing schemes against general adversaries in the asynchronous communication setting. Previously, such schemes were designed in the synchronous communication setting. As an application of our scheme, we present the *first* single round asynchronous perfectly-secure message transmission protocol against general adversaries.

1 Introduction

Secret sharing [Sha79, Bla79] is one of the fundamental problems in distributed cryptography. In its simplest form, it allows a special party D called *dealer* to share a secret among a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n parties. The sharing is done in such a way that certain designated subsets of parties called *access sets* can reconstruct the secret by pooling their shares; on the other hand, subsets of parties which does not constitute an access set get no information about the shared secret. The latter condition holds even if the parties in the non-access sets are computationally unbounded. The set of access sets and non-access sets are represented by Σ and Γ respectively; these sets are called *access structure* and *adversary structure* respectively. It is assumed that there exists a *computationally unbounded* adversary $\mathcal{A}$, who selects a set from Γ for corruption and *passively* corrupts the parties in that set.

Error-decodable secret-sharing (EDSS) is a special type of secret sharing, which allows *robust* reconstruction of the secret, even in the presence of a malicious $\mathcal{A}$. More specifically, it ensures that the honest parties reconstruct the correct secret even if the corrupted parties produce incorrect shares during the reconstruction process. Such schemes are more practically relevant because in practice it is a very strong assumption that adversary will do only passive corruption. A popular adversary structure which is widely studied in the literature is the *threshold* adversary structure, where it is assumed that $\mathcal{A}$ can corrupt at most t parties out of the n parties; for such an $\mathcal{A}$, the set Γ is the set of all possible $\binom{n}{t}$ subsets of t parties. It is well known that EDSS against a threshold adversary is possible if and only if $t < n/3$ [MS81].

A non-threshold adversary is a generalization of threshold adversary, where Γ consists of subsets of arbitrary size. The motivation for studying non-threshold adversaries is that in certain scenarios, threshold adversaries may be un-realistic.

S. Jajodia and C. Mazumdar (Eds.): ICISS 2015, LNCS 9478, pp. 364–377, 2015.
DOI: 10.1007/978-3-319-26961-0_22

For example, certain computer viruses, such as the ILOVEYOU [Mis00] virus and the Internet virus/worm [ER89] spreads only to Windows and Unix respectively. An attacker who can exploit a weakness in one platform, can with al most the same ease attack many computers, if not all, on that same platform. Such a scenario is more naturally captured by a non-threshold adversary instead of a threshold adversary.

EDSS against non-threshold adversaries are first studied in [Kur11], where it is shown that EDSS is possible if and only if $\mathcal{P}$ satisfies $\mathcal{Q}^{(3)}$ condition with respect to Γ (see Definition 4); informally this means that the union of every *three* sets from Γ is a proper subset[1] of $\mathcal{P}$. The sufficiency is shown by designing an efficient EDSS scheme for $\mathcal{Q}^{(3)}$ adversary structures, whose complexity is polynomial in n and the size of the underlying monotone span program (MSP) realizing Γ (see the next section for the definition of MSP).

Our Motivation and Results: All the results discussed above are in the synchronous communication setting, where it is assumed that the parties are synchronized via a global clock and hence there exists strict upper bound on message delays. Unfortunately, real-world networks like the Internet does not provide such synchronization and the messages can be arbitrarily delayed. Motivated by this, [BCG93] introduced the asynchronous communication setting, where the messages can be arbitrarily delayed. Compared to the protocols in the synchronous setting, asynchronous protocol are highly complex. This stems from the fact that in a completely asynchronous protocol, it is impossible to distinguish between a slow but *honest* sender, whose messages are delayed arbitrarily and a *corrupted* sender, who does not send any message at all. As a result, at any stage of the protocol, no party can afford to listen from all the n parties, as this may turn out to be endless. Hence as soon as a party receives communication from "sufficient" number of parties[2], it has to proceed to the next stage, ignoring the communication from the rest of the parties. However, the ignored parties may be potentially honest parties.

Even though the asynchronous communication model is practically relevant, to the best of our knowledge nothing is known in the literature about EDSS in the asynchronous setting. Motivated by this, in this work we initiate the study of asynchronous EDSS (AEDSS). Specifically, we design an AEDSS scheme for $\mathcal{Q}^{(3)}$ adversary structures. The scheme is obtained by modifying the EDSS scheme of [Kur11] to adapt it to the asynchronous setting. Interestingly, the adaptation is not straight forward and requires the parties to iteratively perform certain steps in an "online" fashion upon the disclosure of every share, to deal with the asynchronous nature of the communication (more on this later).

EDSS is very closely related to one round *perfectly-secure message transmission* (PSMT) [Kur11]. On a high level, a PSMT protocol allows a distributed sender and a receiver to carry out reliable and secure communication

[1] Note that this is a generalization of the condition $t < n/3$ for the threshold setting.

[2] For example, in the threshold setting, a party has to proceed to the next step after listening from $n - t$ parties, as t corrupted parties may decide not to send any communication.

over n channels, some of which may be under the control of a computationally unbounded malicious adversary [DDWY93]. Given an EDSS scheme, one can easily design a one round PSMT protocol. As an application of our AEDSS, we present the first one round asynchronous PSMT (APSMT) protocol tolerating a generalized adversary. This significantly improves upon the previous best APSMT protocol against non-threshold adversary [SKR02], which requires $\mathcal{O}(|\mathcal{A}|)$ rounds of interaction between the sender and the receiver; here $|\mathcal{A}|$ denotes the cardinality of the set of channels corrupted by the adversary in the protocol.

Overview of Our AEDSS: We follow [Kur11] and first design a weaker primitive called *weak secret sharing* (WSS). Informally a WSS scheme is similar to EDSS, where sharing is done with respect to a designated party, say P_i, who is given the shared secret as well as the randomness used to compute the shares of the secret. The reconstruction protocol is now invoked by this designated P_i, who publicly reveals the assigned secret and the randomness, which is then compared with the shares disclosed by the individual share holders. If P_i is honest, then it correctly discloses the secret and the randomness. And this will be "consistent" with the individual shares of all but the parties belonging to an access set. Based on this observation, the parties can accordingly decide to accept or reject the secret disclosed by the designated P_i. It is ensured that a secret is accepted if and only if the secret (and the associated randomness) are revealed correctly by the designated P_i [Kur11]. Given a WSS scheme, [Kur11] designed their EDSS as follows: the dealer first computes the shares of the secret and the shares are distributed to the respective share holders. In addition, each individual share is further shared via WSS, where the randomness used for WSS is assigned to the corresponding share holder. Later during the reconstruction phase, each share holder reveals its share via the reconstruction algorithm of the WSS. The properties of WSS ensure that only the correctly revealed shares are accepted, which ensures robust reconstruction.

To design our AEDSS, we first extend the WSS of [Kur11] to the asynchronous setting. Here we need to deal with two issues due to the asynchronous nature of the communication. First, the designated P_i with respect to which WSS is executed may not invoke the reconstruction algorithm if P_i is *corrupted*. As a result, the reconstruction protocol of WSS may not terminate for a corrupted P_i. Second, even if the designated P_i invokes the reconstruction protocol by revealing the secret and the randomness, the individual shares of the share holders will be revealed asynchronously and hence may not be available simultaneously. As a result, each time a new share is revealed, the parties need to verify the consistency of the revealed secret and the corresponding shares in an online fashion. So unlike the synchronous setting, the reconstruction protocol of the asynchronous WSS will no longer be a single step process, but rather an iterative process. Once we have an asynchronous WSS, we obtain AEDSS as follows: the dealer first computes the shares of the secret and each share is further shared via our AWSS. During the reconstruction phase, each share holder reveals its share by executing the reconstruction protocol of the AWSS. However

due to the asynchronous nature of the communication, the parties cannot afford to terminate the reconstruction of all the AWSS instances. Hence as soon as the parties terminate the AWSS instances of a set of parties belonging to an access set, they reconstruct the secret by using the shares revealed in those instances.

2 Preliminaries

We assume a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n parties and an *external* dealer $\mathsf{D} \notin \mathcal{P}$. The parties are connected by pair-wise private and authentic channels. There exists a finite field $\mathbb{F}$ and all computation are assumed to be performed over $\mathbb{F}$. We denote by $\mathbf{S}$ the set of all possible secrets that can be shared. The distrust among the parties is modeled by a centralized adversary $\mathcal{A}$, who can corrupt a subset of parties from $\mathcal{P}$. The set of potential subsets of corruptible parties is denoted by an *adversary structure* Γ, where $\Gamma \subseteq 2^{\mathcal{P}}$. Adversary $\mathcal{A}$ is assumed to be *static*, who decides the subset of parties to corrupt at the beginning of the execution of a protocol; the corrupted subset is one of the elements of Γ. The adversary is computationally unbounded and can force the corrupted parties to deviate from the protocol instructions in any arbitrary fashion. The dealer D is always assumed to be honest. Let $\Sigma = \Gamma^c$, where Γ^c denotes the complement of Γ; we call the set Σ *access structure* and the elements in Σ are called *access sets*. We next define secret-sharing scheme.

Secret-Sharing Scheme: In a secret-sharing scheme, D has a secret which it wants to share among $\mathcal{P}$. The sharing needs to be done in a way that the parties in any access set can reconstruct the secret by combining their shares, while the parties in any set belonging to the adversary structure gets no information about the secret. More formally:

Definition 1 (Secret-Sharing Scheme [Kur11]). *A secret-sharing scheme over the adversary structure Γ and access structure $\Sigma = \Gamma^c$ is a pair of algorithms $(\mathsf{Sh}, \mathsf{Rec})$ where:*

- $(\mathsf{Share}_1, \ldots, \mathsf{Share}_n) \leftarrow \mathsf{Sh}(s, r)$*: the sharing algorithm* Sh *takes the secret s to be shared along with some randomness r and computes the shares* $\mathsf{Share}_1, \ldots, \mathsf{Share}_n$*, with* Share_i *designated for the party P_i, for $i = 1, \ldots, n$.*
- Rec *is the reconstruction algorithm such that:*

$$\mathsf{Rec}(A, \mathsf{Share}_A) = \begin{cases} s, & if \quad A \in \Sigma \\ \perp, & if \quad A \in \Gamma \end{cases}$$

  where $\mathsf{Share}_A = \{\mathsf{Share}_i | P_i \in A\}$.
- *The following holds:*

$$H(S|SHARE_A) = \begin{cases} 0, & if \quad A \in \Sigma \\ H(S), & if \quad A \in \Gamma \end{cases}$$

  where H is the entropy function [CT06], S is a random variable induced by s and $SHARE_A$ is a random variable induced by Share_A.

We next define monotone access structures.

Definition 2 (Monotone Access Structure). *An access structure Σ is called monotone provided the following holds:*

$$if \;\; A \in \Sigma \;\; and \;\; A' \supseteq A, \;\; then \;\; A' \in \Sigma.$$

In [CDM00] it is shown how to design a secret-sharing scheme for any given monotone access structure Σ using monotone span programs (MSP); we briefly recall the same in the sequel.

Linear Secret-Sharing Scheme (LSSS) and MSP [CDM00]**:** On a very high level, an MSP M is an $\ell \times d$ matrix over $\mathbb{F}$, with $\ell \geq d$ and $\ell \geq n$, where

$$M = \begin{pmatrix} m_1 \\ \vdots \\ m_\ell \end{pmatrix}.$$

There exists a labeling function $\psi : \{1, \ldots, \ell\} \rightarrow \mathcal{P}$ and we say row j is *associated* with party P_i if $\psi(j) = P_i$. For a subset of parties $A \subseteq \mathcal{P}$, let M_A be the submatrix of M consisting of the rows m_i such that $\psi(i) \in A$. Matrix M has the property that A is an access set if and only if the vector $(1, 0, \ldots, 0)$ is in the linear span of M_A. Given such an M, an LSSS can be designed as follows:

Algorithm Sh**:** To share a secret $s \in \mathbb{F}$, the dealer D does the following:

- Select a random vector $r \in \mathbb{F}^{d-1}$ and compute a vector

$$v = M \times \begin{pmatrix} s \\ r \end{pmatrix}$$

  where $v = (v_1, \ldots, v_\ell)^T$.
- Let $\mathsf{LSSS}(s, r) \stackrel{def}{=} (\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$, where $\mathsf{Share}_i = \{v_j | \psi(j) = P_i\}$. Dealer D gives Share_i to party P_i for $i = 1, \ldots, n$.

Algorithm Rec**:** Let $A \in \Sigma$ be an access set. To reconstruct s, the parties in A do the following:

- Let μ_A be a row vector such that $\mu_A \cdot M_A = (1, 0, \ldots, 0)$; such a μ_A is bound to exist as $(1, 0, \ldots, 0)$ is in the linear span of M_A. Given such a μ_A, the parties in A reconstruct s by computing:

$$s = \mu_A \cdot \mathsf{Share}_A, \;\; \text{where} \;\; \mathsf{Share}_A = \{\mathsf{Share}_i | P_i \in A\}.$$

We say that the above (M, ψ) is an MSP which realizes[3] Γ. In [CDM00] it is shown how to design an MSP realizing any monotone access structure. Moreover, it is also shown that the above pair of algorithms $(\mathsf{Sh}, \mathsf{Rec})$ indeed constitute a valid secret-sharing scheme.

[3] Readers familiar with the classical (n, t) Shamir secret-sharing scheme [Sha79] can see that M for the Shamir's scheme is the $n \times (t + 1)$ Vandermonde matrix. The vector $(s, r)^T$ constitutes the coefficients of the sharing polynomial of degree at most t, with s as the constant term. The reconstruction vector μ_A consists of the Lagrange's reconstruction coefficients.

In our protocols, we use the following metric to check the locations at which two vectors of shares match.

Definition 3. *Let* $\mathsf{Share} = (\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$ *and* $\mathsf{Share}^\star = (\mathsf{Share}_1^\star, \ldots, \mathsf{Share}_n^\star)$ *be two vectors of shares, where* $\mathsf{Share}_i, \mathsf{Share}_i^\star$ *are associated with party* P_i*, for* $i = 1, \ldots, n$*. Then* $\mathsf{Match}(\mathsf{Share}, \mathsf{Share}^\star) \stackrel{def}{=} \{P_i | \mathsf{Share}_i = \mathsf{Share}_i^\star\}$.

In our protocol, we will use the following property of LSSS, which simply follows from the property of MSP that the shares of the parties in an access set uniquely determine the shared secret.

Lemma 1 ([CDM00,Kur11]). *Let* Share *and* $\mathsf{Share}^\star$ *be two vectors of shares, where* $\mathsf{Share} = \mathsf{LSSS}(s, r), \mathsf{Share}^\star = \mathsf{LSSS}(s^\star, r^\star)$ *and* Σ *is the underlying access structure. If* $\mathsf{Match}(\mathsf{Share}, \mathsf{Share}^\star) \in \Sigma$*, then* $s = s^\star$.

In our protocols, we often require to verify whether a given set of parties A is an access set. This can be done in time polynomial in the size of the underlying MSP by verifying whether the row vector $(1, 0, \ldots, 0)$ is in the linear span of M_A. We next present the following definition of $\mathcal{Q}^{(k)}$ condition from [HM97].

Definition 4 ($\mathcal{Q}^{(k)}$ Condition [HM97]). *Let* $\mathcal{S} \subseteq \mathcal{P}$ *be a set and* Γ *be an adversary structure over* $\mathcal{P}$*. We say that* $\mathcal{S}$ *satisfies* $\mathcal{Q}^{(k)}$ *condition with respect to* Γ *if there exists no* k *sets* $\mathcal{B}_1, \ldots, \mathcal{B}_k \in \Gamma$*, such that* $\mathcal{S} \subseteq \mathcal{B}_1 \cup \ldots \cup \mathcal{B}_k$.

Finally we note that like the standard secret-sharing schemes, we assume a fixed set of n parties. However it is well known in the literature how to deal with situations where the set of parties changes dynamically (see for example [NS13]); similar techniques are applicable even against generalized adversary.

2.1 The Asynchronous Model and Definitions

Our protocols are designed in the asynchronous communication setting, where there exists no global clock and the channels between the parties have arbitrary delays; thus there are no strict upper bounds within which messages reach to their destinations. The only guarantee in this model is that the messages sent by the honest parties will eventually reach to their destinations. The order of the message delivery is decided by a *scheduler*. To model the worst case scenario, we assume that the scheduler is under the control of the adversary. The scheduler can only schedule the messages exchanged between the honest parties, without having access to the "contents" of these messages. We consider a protocol execution in the asynchronous setting as a sequence of *atomic steps*, where a single party is *active* in each such step. A party is activated when it receives a message. On receiving a message, it performs an internal computation and then possibly sends messages on its outgoing channels. The order of the atomic steps are controlled by the scheduler. At the beginning of the computation, each party will be in a special *start* state. A party is said to *terminate/complete* the computation if it reaches a *halt* state, after which it does not perform any further computation. A protocol execution is said to be complete if all the honest parties terminate

the computation. For an excellent introduction to the asynchronous protocols, see [Can95].

We next define asynchronous error-decodable secret-sharing scheme (AEDSS). Informally such a scheme consists of two protocols, a sharing protocol and a reconstruction protocol. The sharing protocol allows the dealer D to share a secret among $\mathcal{P}$. The reconstruction protocol allows the parties to reconstruct the shared secret, even if the corrupted parties provide incorrect shares. Both the protocols terminate for the honest parties. Formally:

Definition 5 (AEDSS). *Let* $(\mathsf{AEDSS\text{-}Sh}, \mathsf{AEDSS\text{-}Rec})$ *be a pair of asynchronous protocols for the dealer* D *and the set of parties* $\mathcal{P}$*. Dealer* D *has a private input* $s \in \mathbb{F}$ *for the protocol* $\mathsf{AEDSS\text{-}Sh}$*, which it wants to share among* $\mathcal{P}$*. Then* $(\mathsf{AEDSS\text{-}Sh}, \mathsf{AEDSS\text{-}Rec})$ *is called an AEDS scheme for the adversary structure* Γ *if the following are satisfied for every possible* $\mathcal{A}$*:*

- **Termination**. *Every honest party eventually terminates* $\mathsf{AEDSS\text{-}Sh}$, $\mathsf{AEDSS\text{-}Rec}$.
- **Correctness**. *Every honest party upon terminating* $\mathsf{AEDSS\text{-}Rec}$ *outputs* s.
- **Privacy**. *No information about* s *is revealed to* $\mathcal{A}$ *during* $\mathsf{AEDSS\text{-}Sh}$.

To design our AEDSS, we actually require a weaker primitive called *asynchronous weak secret-sharing* (AWSS). Like AEDSS, an AWSS scheme also consists of a sharing protocol and a reconstruction protocol. During the sharing protocol, D shares a secret s among $\mathcal{P}$; additionally the secret s is also handed over to a *designated* party $P_i \in \mathcal{P}$. The reconstruction protocol allows P_i to reveal s to a *designated* party $P_R \in \mathcal{P}$. The sharing protocol always terminate for the honest parties. But the reconstruction protocol need not always terminate for P_i and P_R; however it always terminates if P_i and P_R are *honest*. Moreover, it is required that if an *honest* P_R terminates the reconstruction protocol, then the reconstructed value is the same as distributed by D to P_i. More formally:

Definition 6 (AWSS). *Let* $(\mathsf{AWSS\text{-}Sh}, \mathsf{AWSS\text{-}Rec})$ *be a pair of asynchronous protocols for a pair of designated parties* $P_i, P_R \in \mathcal{P}$*, the dealer* D *and the set of parties* $\mathcal{P}$*. Dealer* D *has a private input* $s \in \mathbb{F}$ *for the protocol* $\mathsf{AWSS\text{-}Sh}$*, which it wants to give to* P_i *and share it among* $\mathcal{P}$*. Party* P_i *has a private input* $s^\star$ *for* $\mathsf{AWSS\text{-}Rec}$*, which it wants to reveal to party* P_R*. Then* $(\mathsf{AWSS\text{-}Sh}, \mathsf{AWSS\text{-}Rec})$ *is called an AWSS scheme for the adversary structure* Γ *if the following are satisfied for every possible* $\mathcal{A}$*:*

- **Termination**. *All the following should be satisfied:*
  - *Every honest party eventually terminates* $\mathsf{AWSS\text{-}Sh}$.
  - *Every honest party in* $\mathcal{P} \setminus \{P_i, P_R\}$ *eventually terminates* $\mathsf{AWSS\text{-}Rec}$*. Moreover, if* P_i *and* P_R *are honest then they also eventually terminate* $\mathsf{AWSS\text{-}Rec}$.
- **Correctness**. *The following holds:*
  - *If* P_i *is honest then it obtains* s *at the end of* $\mathsf{AWSS\text{-}Sh}$.
  - *If* P_R *is honest and terminates* $\mathsf{AWSS\text{-}Rec}$*, then* $s^\star = s$.
- **Privacy**. *If* P_i *is honest then no information about* s *is revealed during* $\mathsf{AWSS\text{-}Sh}$.
  - *If* P_R *is honest then no information about* s *is revealed during* $\mathsf{AWSS\text{-}Rec}$.

3 Asynchronous Weak Secret-Sharing Scheme (AWSS)

Let $\mathcal{P}$ satisfies $\mathcal{Q}^{(3)}$ condition with respect to Γ. We present an AWSS scheme for Γ. The AWSS scheme consisting of protocols AWSS-Sh (for the sharing phase) and AWSS-Rec (for the reconstruction of the secret by a designated party) is presented in Fig. 1. Protocol AWSS-Sh is straight forward: let (M, ψ) be an MSP realizing Γ, where M is of size $\ell \times d$. The dealer then computes the shares according to the LSSS and distributes it among the parties. In addition, the secret along with the randomness used in the LSSS are handed to the designated party P_i. The protocol eventually terminates for every honest party.

During AWSS-Rec, party P_i first reveals the secret along with the randomness to the designated party P_R. Hence the participation of P_i is very crucial for the termination of AWSS-Rec; a corrupted P_i may choose not to participate in the protocol, in which case the protocol does not terminate for P_R. Independently, every party hands over their shares to P_R. Party P_R on receiving the secret and randomness from P_i, itself computes the shares of all the parties according to the LSSS. It then matches these shares with the ones it received from the corresponding parties. The comparison is performed till the matching occurs for all but a set of parties belonging to the adversary structure. This ensures that the matching occurs for a set of parties satisfying $\mathcal{Q}^{(2)}$ condition. Note that the shares of the parties reach asynchronously to P_R. Hence P_R needs to perform the comparison every time it receives a new share. The idea here is that the set of *honest* parties in $\mathcal{P}$ satisfy $\mathcal{Q}^{(2)}$ condition and their shares will eventually reach to P_R. Moreover, if P_i is *honest*, it correctly reveals the secret and randomness to P_R; so eventually the shares sent by the honest parties will match with the corresponding shares, computed by P_R itself from the revealed secret and the randomness. On the other hand, if P_i is *corrupted* and the matched set satisfies $\mathcal{Q}^{(2)}$ condition, then also it is ensured that P_i has revealed the correct secret. This is because among these matched set of parties, the set of honest parties will constitute an *access* set, whose shares uniquely determine the original secret.

The properties of AWSS-Sh and AWSS-Rec are stated in Theorem 1.

Theorem 1. *Let $\mathcal{A}$ be an adversary specified by an adversary structure Γ over $\mathcal{P}$, such that $\mathcal{P}$ satisfies $\mathcal{Q}^{(3)}$ condition with respect to Γ. Then* (AWSS-Sh, AWSS-Rec) *constitutes a valid AWSS scheme for Γ. Protocol* AWSS-Sh *runs in time polynomial in $|\mathbf{S}|$ and ℓ. Protocol* AWSS-Rec *runs in time polynomial in $|\mathbf{S}|, \ell$ and n.*

Proof **(Termination).** Since D is honest, protocol AWSS-Sh eventually terminates for every honest party. During AWSS-Rec, every honest party in the set $\mathcal{P} \setminus \{P_i, P_R\}$ terminates after sending its share to P_R. Next we consider an honest P_i and P_R. If P_i is honest, then P_R eventually receives (s, r) from P_i. Moreover, the set of honest parties in $\mathcal{P}$ satisfies $\mathcal{Q}^{(2)}$ condition with respect to Γ. Furthermore, the shares of each honest party eventually reaches P_R. Given this, it is easy to see that P_R eventually finds that the set $\mathcal{P} \setminus \mathsf{Match}(\mathbf{Y}, \mathbf{Y}') \in \Gamma$ and terminates.

Algorithm AWSS-Sh(D, P_i, s)

DISTRIBUTING THE SHARES, SECRET AND THE RANDOMNESS — Only D executes the following code:

- On having a secret $s \in \mathbb{F}$, D randomly selects $r \in \mathbb{F}^{d-1}$, computes $\mathsf{LSSS}(s, r) = (\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$.
- D sends (s, r) to party P_i.
- D sends Share_j to every party $P_j \in \mathcal{P}$ as its share and terminate.

RECEIVING THE SHARES — Every party $P_j \in \mathcal{P}$ executes the following code:

- Wait to receive the share Share_j from D. If $P_j = P_i$ then additionally wait to receive (s, r). On receiving, terminate.

Algorithm AWSS-Rec(P_i, P_R, s)

DISCLOSING THE SECRET AND THE RANDOMNESS — Only P_i executes the following code:

- Send (s, r) to the party P_R.

DISCLOSING SHARES — Every $P_j \in \mathcal{P}$ (including P_i and P_R) executes the following code:

- Send the share Share_j to the party P_R. If $P_j \neq P_R$ then terminate upon sending.

RECONSTRUCTING THE SECRET — Only party P_R executes the following code:

- Wait to receive (s, r) from P_i. Upon receiving, compute $\mathsf{LSSS}(s, r) = (\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$. Let $\mathbf{Y} = (\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$
- Create a vector $\mathbf{Y}'$ of length n, initialize it to $(\perp, \ldots, \perp(n \text{ times}))$; here $\perp$ is a special symbol not in $\mathbb{F}$. Repeat the following after receiving a share Share_j from any party P_j such that $\mathbf{Y}'(j) = \perp$:
  - Set $\mathbf{Y}'(j) = \mathsf{Share}_j$.
  - Check if $\mathcal{P} \setminus \mathsf{Match}(\mathbf{Y}, \mathbf{Y}') \in \Gamma$.
  - If the above condition is satisfied then output s and terminate.

Fig. 1. Asynchronous weak secret-sharing scheme

Correctness. We have to consider an honest P_R. Since P_R terminates, it implies that $\mathcal{P} \setminus \mathsf{Match}(\mathbf{Y}, \mathbf{Y}') \in \Gamma$. This further implies that $\mathsf{Match}(\mathbf{Y}, \mathbf{Y}')$ satisfies $\mathcal{Q}^{(2)}$ condition with respect to Γ. If not, then the set $\mathsf{Match}(\mathbf{Y}, \mathbf{Y}') \cup \mathcal{P} \setminus \mathsf{Match}(\mathbf{Y}, \mathbf{Y}') = \mathcal{P}$ fails to satisfy $\mathcal{Q}^{(3)}$ condition, which is a contradiction. Let P_R receives $(s^\star, r^\star)$ from P_i, implying $\mathbf{Y} = \mathsf{LSSS}(s^\star, r^\star)$. Note that if P_i is honest then $(s^\star, r^\star) = (s, r)$. Let $\mathsf{Com} = \mathsf{Match}(\mathbf{Y}, \mathbf{Y}')$ and let Com-Hon be the set of honest parties in the set Com. It is easy to see that Com-Hon is an access set, as otherwise this will contradict the fact that Com satisfies $\mathcal{Q}^{(2)}$. Now this implies that $\mathsf{LSSS}(s, r)$ and $\mathsf{LSSS}(s^\star, r^\star)$ are the same, with respect to the parties in Com-Hon. This from Lemma 1 implies that $s^\star = s$.

Privacy. During AWSS-Sh, the dealer D just distributes the shares computed according to LSSS and there is no interaction among the parties. So it follows from the properties of LSSS that if P_i is honest, then no information about s is

revealed to $\mathcal{A}$. During AWSS-Rec, all the shares are sent only to P_R, along with (s, r). So if P_R is honest, then the privacy of s is preserved.

Efficiency. During AWSS-Sh, computing the shares costs time polynomial in $|\mathbf{S}|$ and ℓ for D. During AWSS-Rec, party P_R has to verify if the set $\mathcal{P} \setminus \mathsf{Match}(\mathbf{Y}, \mathbf{Y}') \in \Gamma$; moreover this verification may need to be performed n times in the worst case. So overall this costs time polynomial in n and ℓ for P_R. □

4 Asynchronous Error-Decodable Secret-Sharing Scheme (AEDSS)

Let $\mathcal{P}$ satisfy $\mathcal{Q}^{(3)}$ condition with respect to Γ and let (M, ψ) be an MSP realizing Γ, where M is of size $\ell \times d$; we present an AEDSS tolerating $\mathcal{A}$. Note that $\mathcal{P}$ satisfying $\mathcal{Q}^{(3)}$ is a necessary condition for the existence of EDSS even in the synchronous communication setting. So obviously it is necessary even for AEDSS. The AEDSS scheme consisting of protocols AEDSS-Sh (for the sharing phase) and AEDSS-Rec (for the public reconstruction of the secret) is presented in Fig. 2. For simplicity and without loss of generality, we assume that $\ell = n$ and $\psi(i) = i$ for $i = 1, \ldots, n$.

During AEDSS-Sh, the dealer first computes the shares of the secret according to the LSSS and distributes the shares among the parties. In addition, for each share, it executes an instance of AWSS-Sh to further share the share; as a result, each party will have a share of each share. The protocol eventually terminates for the honest parties. During AEDSS-Rec, each share holder P_j executes an instance $\mathsf{AWSS\text{-}Rec}_{ji}$ of AWSS-Rec to reveal its share to every other party P_i. Party P_i waits to terminate AWSS-Rec instances corresponding to the parties in an access set. Once it terminates those many instances, it reconstructs the secret using the shares revealed at the end of those instances. The idea here is that the instances $\mathsf{AWSS\text{-}Rec}_{ji}$ executed by each *honest* P_j eventually terminates for each *honest* P_i and the set of honest parties constitute an access set. Moreover, for every instance $\mathsf{AWSS\text{-}Rec}_{ji}$ terminated by P_i, the share revealed by P_j is the same as distributed by the dealer; this is true even if P_j is corrupted (follows from the properties of AWSS-Rec). So every honest P_i eventually terminates the protocol with the correct secret.

The properties of AEDSS-Sh and AEDSS-Rec are stated in Theorem 2.

Theorem 2. *Let $\mathcal{A}$ be an adversary specified by an adversary structure Γ over $\mathcal{P}$, such that $\mathcal{P}$ satisfies $\mathcal{Q}^{(3)}$ condition with respect to Γ. Then* (AEDSS-Sh, AEDSS-Rec) *constitutes a valid AEDSS for Γ. Both protocols run in time polynomial in $|\mathbf{S}|, n$ and ℓ.*

Proof **(Termination).** Since D is honest, the instances $\mathsf{AWSS\text{-}Sh}_1, \ldots, \mathsf{AWSS\text{-}Sh}_n$ eventually terminates for every honest party and so every honest party eventually terminates AEDSS-Sh. We next claim that AEDSS-Rec also terminates eventually for every honest party P_i. This follows from the fact that the $\mathsf{AWSS\text{-}Rec}_{ji}$ instances invoked by *honest* parties P_j corresponding to P_i eventually terminates (follows from Theorem 1) and the set of honest parties constitutes an access set.

Algorithm AEDSS-Sh(D, s)

DISTRIBUTING THE SHARES — Only D executes the following code:

- On having a secret $s \in \mathbb{F}$, D randomly selects $r \in \mathbb{F}^{d-1}$ and computes $\mathsf{LSSS}(s, r) = (\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$.
- For every $P_i \in \mathcal{P}$, D invokes an instance AWSS-Sh$(\mathsf{D}, P_i, \mathsf{Share}_i)$ of AWSS-Sh. Denote this instance of AWSS-Sh as AWSS-Sh$_i$. Let r_i be the randomness used during AWSS-Sh$_i$ and $\mathsf{Share}_{i1}, \ldots, \mathsf{Share}_{in}$ be the shares of the share Share_i generated during AWSS-Sh$_i$.
- Terminate after terminating the instances AWSS-Sh$_1, \ldots,$ AWSS-Sh$_n$.

RECEIVING THE SHARE INFORMATION — Every party $P_j \in \mathcal{P}$ executes the following code:

- Corresponding to every $P_i \in \mathcal{P}$, participate in the instance AWSS-Sh$_i$ invoked by D.
- Terminate after terminating the instances AWSS-Sh$_1, \ldots,$ AWSS-Sh$_n$.

Let $(\mathsf{Share}_j, r_j), \mathsf{Share}_{jj}$ be obtained at the end of AWSS-Sh$_j$ and Share_{ij} be obtained at the end of AWSS-Sh$_i$ for every $P_i \in \mathcal{P} \setminus P_j$.

Algorithm AEDSS-Rec(s)

DISCLOSING THE SHARES — Every party $P_j \in \mathcal{P}$ executes the following code:

- Corresponding to every $P_i \in \mathcal{P}$, invoke an instance AWSS-Rec$(P_j, P_i, \mathsf{Share}_j)$ of AWSS-Rec by sending (Share_j, r_j) to party P_j. Denote this instance of AWSS-Rec as AWSS-Rec$_{ji}$.
- Corresponding to every $P_k, P_i \in \mathcal{P}$, participate in the instance AWSS-Rec$_{ki}$ of AWSS-Rec by sending the share-share Share_{kj} to the party P_i.

RECONSTRUCTING THE SECRET AND TERMINATION — Every party $P_i \in \mathcal{P}$ executes the following code:

- Create a set $\mathcal{C}_i$, which is initially $\emptyset$. Include party P_j to $\mathcal{C}_i$ if P_i terminates the instance AWSS-Rec$_{ji}$ with output Share_j.
- Wait till $\mathcal{C}_i \in \Sigma$. Once $\mathcal{C}_i \in \Sigma$, reconstruct s by applying the reconstruction algorithm of the LSSS to the shares $\{\mathsf{Share}_j\}_{P_j \in \mathcal{C}_i}$ and terminate.

Fig. 2. Asynchronous error-decodable secret-sharing scheme

Correctness. Let P_i be an *honest* party. For correctness, we need to argue that if $P_j \in \mathcal{C}_i$ then Share_j obtained at the end of the instance AWSS-Rec$_{ji}$ is indeed correct. However, this follows from the correctness property of AWSS-Rec (follows from Theorem 1).

Privacy. During AEDSS-Sh, the adversary gets no information about the shares of the honest parties, as they are shared via AWSS; this follows from the privacy property of AWSS. Given this, it is easy to see that s remains private during AEDSS-Sh.

Efficiency. In the protocol, n instances of AWSS-Sh and n^2 instances of AWSS-Rec are executed. It now follows easily that both AEDSS-Sh and AEDSS-Rec runs in time polynomial in $|\mathbf{S}|, n$ and ℓ. □

Notation 1. *In the next section, while using* AEDSS-Sh *and* AEDSS-Rec *we will use the following notation:*

- $\mathsf{AEDSS\text{-}Sh}(s, r) = (\widehat{\mathsf{Share}_1}, \ldots, \widehat{\mathsf{Share}_n})$*: this denotes* D *executing* AEDSS-Sh *with secret* s *and randomness* r *and computing all the information to be distributed among the parties. Here* $\widehat{\mathsf{Share}_j}$ *denotes all the information distributed by* D *to the party* P_j*. Thus* $\widehat{\mathsf{Share}_j} = (\mathsf{Share}_j, r_j, \{\mathsf{Share}_{ij}\}_{i=1}^{n})$.
- $\mathsf{AEDSS\text{-}Rec}_i(\cdot) = s$*: this denotes party* P_i *reconstructing* s *by executing its part of the code of* AEDSS-Rec(s)*. This is an online process, where* P_i *asynchronously receives information from various parties and performs computation on them, till it receives sufficient information to reconstruct* s*.*

5 Application of AEDSS to Asynchronous Perfectly-Secure Message Transmission (APSMT)

In the model of *perfectly-secure message transmission* (PSMT), there exists a sender S and a receiver R connected by n channels $\mathcal{W} = \{w_1, \ldots, w_n\}$, some of which may be under the control of a computationally unbounded malicious adversary $\mathcal{A}$. There exists a message $m \in \mathbb{F}$, which S wants to reliably and privately communicate to R over the n channels, even in the presence of the adversary. In [SKR02], asynchronous PSMT (APSMT) is studied in the presence of a non-threshold adversary. In the asynchronous model, the channels are not synchronized and there can be arbitrary delays; the only guarantee is that information sent over honest channels reach to their destination eventually. The non-threshold adversary is characterized by an adversary structure Γ over $\mathcal{W}$, which denotes the set of possible subsets of channels which can be potentially corrupted by $\mathcal{A}$; during the execution of a protocol, adversary can select any subset of channels from Γ for corruption. In [SKR02] it is shown that APSMT tolerating $\mathcal{A}$ is possible if and only if $\mathcal{W}$ satisfies $\mathcal{Q}^{(3)}$ condition with respect to Γ. To prove the sufficiency of the $\mathcal{Q}^{(3)}$ condition, they presented a protocol, which requires $\mathcal{O}(|\mathcal{A}|)$ rounds of interaction between S and R, where $|\mathcal{A}|$ denotes the cardinality of the set of channels corrupted by $\mathcal{A}$ in the protocol. We present an APSMT protocol, which requires only one round of interaction between S and R, thus significantly improving the protocol of [SKR02].

Our APSMT protocol called APSMT (see Fig. 3) is adapted from our AEDSS, where S plays the role of the dealer and $\mathcal{W}$ is treated as $\mathcal{P}$, with w_i playing the "role" of party P_i. Specifically, S considers m as the secret to be shared among $\mathcal{P}$ and computes the information to be distributed among the parties as part of AEDSS-Sh; the information that needs to be given to party P_i is sent over the channel w_i. Receiver R asynchronously receives information over the channels and recovers m by executing the steps of AEDSS-Rec that an honest party would have executed to recover m.

The properties of APSMT are stated in Theorem 3, which simply follow from the protocol steps and the properties of AEDSS-Sh, AEDSS-Rec.

Protocol $\mathsf{APSMT}(\mathsf{S}, m)$

CODE FOR S — Only S executes the following code:

- On having the message m, S selects a random string r and computes $\mathsf{AEDSS\text{-}Sh}(m, r) = (\widehat{\mathsf{Share}_1}, \ldots, \widehat{\mathsf{Share}_n})$.
- For $i = 1, \ldots, n$, S sends $\widehat{\mathsf{Share}_i}$ over channel w_i and terminate.

CODE FOR R — Only R executes the following code:

- R asynchronously receives information over the channels, computes $\mathsf{AEDSS\text{-}Rec}_i(\cdot) = m$ and terminates.

Fig. 3. Single round APSMT protocol

Theorem 3. *Let $\mathcal{A}$ be an adversary specified by an adversary structure Γ over $\mathcal{W}$, such that $\mathcal{W}$ satisfies $\mathcal{Q}^{(3)}$ condition with respect to Γ. Then* APSMT *constitutes a valid APSMT protocol. Protocol* APSMT *runs in polynomial time in $|\mathbf{M}|, \ell$ and n, where $\mathbf{M}$ is the set of all possible messages that can be communicated.*

6 Open Problems

Our AEDSS requires computation time polynomial in the size of the underlying MSP. In the worst case, the underlying MSP may be exponential in n. On the other hand, certain access structures like the threshold access structures have very efficient MSP and hence error-decoding mechanism, requiring computation time polynomial only in n. It is a very interesting open problem to design AEDSS for arbitrary access structures with running time polynomial in n.

Acknowledgement. The author would like to thank the anonymous referees for their useful feedback.

References

[BCG93] Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC, pp. 52–61. ACM (1993)

[Bla79] Blakley, G.R.: Safeguarding cryptographic keys. In: AFIPS National Computer Conference, pp. 313–317 (1979)

[Can95] Canetti, R.: Studies in secure multiparty computation and applications. Ph.D. thesis, Weizmann Institute, Israel (1995)

[CDM00] Cramer, R., Damgård, I.B., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000)

[CT06] Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley, New York (2006)

[DDWY93] Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. J. ACM **40**(1), 17–47 (1993)

[ER89] Eichin, M.W., Rochlis, J.A.: With microscope and tweezers: an analysis of the internet virus of November 1988. In: IEEE Symposium on Security and Privacy, pp. 326–343. IEEE Computer Society (1989)

[HM97] Hirt, M., Maurer, U.M.: Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In: PODC, pp. 25–34. ACM (1997)

[Kur11] Kurosawa, K.: General error decodable secret sharing scheme and its application. IEEE Trans. Inf. Theory **57**(9), 6304–6309 (2011)

[Mis00] Computer bug bites hard, spreads fast [online] (2000). http://www.cnn.com/2000/TECH/computing/05/04/iloveyou.01/index.html

[MS81] McEliece, R.J., Sarwate, D.V.: On sharing secrets and reed-solomon codes. Commun. ACM **24**(9), 583–584 (1981)

[NS13] Nojoumian, M., Stinson, D.R.: On dealer-free dynamic threshold schemes. Adv. Math. Commun. **7**(1), 39–56 (2013)

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

[SKR02] Srinathan, K., Kumar, M.V.N.A., Pandu Rangan, C.: Asynchronous secure communication tolerating mixed adversaries. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 224–242. Springer, Heidelberg (2002)