AS^3: Adaptive Social Secret Sharing for Distributed Storage Systems

Giulia Traverso, Denise Demirel, Sheikh Mahbub Habib, and Johannes Buchmann
TU Darmstadt, Germany

***Abstract*—Distributed storage allows to outsource a document to the cloud such that multiple users can easily access the file. The protection of the document stored relies on secret sharing, which generates and distributes shares of the document to the storage servers. However, the users have to trust that a certain amount of storage servers behaves honestly and do not lose (retrievability) or reveal (confidentiality) the document. To address this so called social secret sharing schemes were developed that allow to adjust the distribution of shares according to the experience made with the involved storage servers. In this work, we provide a framework called AS^3 that allows to build social secret sharing schemes based on dynamic secret sharing. The resulting protocol has more freedom in adjusting the parameters of the shares distribution and therefore leads to more efficient and accurate solutions as well as an optimal storage consumption. Furthermore, we provide measures to detect and to prevent that the document is lost or accidentally revealed to individual storage servers. We also demonstrate how to compute trust values for storage servers, how to initialize trust values for newcomers, and provide a proof of concept implementation.**

Keywords: distributed storage, social secret sharing, applied cryptography, trust, dynamic secret sharing.

I. Introduction

Data storage is one of the main services offered in cloud computing to consumers. In fact, when document owners do not have the resources to store large amounts of data, they outsource these data to the storage servers of one or multiple commercial cloud providers, e.g. Amazon, Google, or Microsoft. There are two crucial aspects that cloud providers must ensure: *confidentiality* and *retrievability*. Confidentiality means that the data remain private and cannot be accessed by malicious third parties, including the storage providers themselves. Retrievability refers to protecting the data from being lost and provide access to it on demand and in a reasonable amount of time.

Distributed storage systems are an interesting solution for data storage, which does not involve encryption and thus does not introduce challenges with respect to key management nor weaknesses with respect to unbound attackers. More precisely, suppose a user wants to outsource a document to the cloud. Then, the document is distributed across a set of n storage servers from one or multiple cloud providers using *secret sharing* [18], [1]. Secret sharing is a cryptographic primitive that provides both confidentiality and retrievability. Confidentiality is achieved because the document is shared such that any subset of at least $k \leq n$ storage servers holding shares can reconstruct the document, meaning that any subset of $k-1$ (or less) colluding storage servers cannot get any information about the document. Retrievability is achieved because only k out of n shares are necessary for the document reconstruction, meaning that the user can access the document stored even if up to $n-k$ storage servers have a breakdown. Furthermore, secret sharing does not rely on computational hardness assumptions and, thus, is not prone to computationally unbounded attackers.

Some cloud providers, and thus their storage servers, might be more trustworthy to keep the shares stored confidential and available than others. Clearly, one prefers that the less trustworthy a cloud provider is, the less reconstruction power it should get. Note that the higher the reconstruction power of a storage server, the less other storage servers are needed in the reconstruction of the document stored. This is realized by *social secret sharing*, [11], [13], [12], [14], where the distributed storage system is equipped with a trust function that keeps the behavior of the storage servers monitored. Using this technique, shares can be distributed to cloud providers considering their reputation with respect to confidentiality and retrievability. However, current solutions have severe shortcomings. First, certain details have not been elaborated yet, e.g. how to initialize parameters and how the trust function affects the distribution of shares. Second, they do not provide measures to detect nor to sufficiently prevent "instable states", i.e. states where the shares are distributed in a way that the document is lost or revealed to one single storage server. Third, storage space consumptions and computational overhead are not optimal. The parameters selection is done once and for all and therefore the parameters cannot be chosen tight enough to ensure optimal efficiency throughout the entire lifetime of the document storage. Fourth, the existing approaches employ trust functions measuring the behavior of the storage servers with respect to retrievability only, leaving the confidentiality aspect uncovered.

In Section II, we discuss different types of secret sharing techniques and their basic characteristics. In Section III, we introduce AS^3, a framework for adaptive social secret sharing schemes that overcomes all the shortcomings mentioned above. More precisely, we define how to initialize the parameters of AS^3 and how the distribution of shares should be adapted according to the measured trust values. Furthermore, we provide rules that allow to detect and react to instable states. In Section IV, we provide details about how to compute trust values using well established trust models and how to initialize trust values for newcomers. Here, both behaviors with respect to confidentiality and retrievability are taken into account. In Section V, we compare our solutions with the existing approaches and show that AS^3 not only makes instable states unlikely, but also leads to a scheme that produces less overhead and consumes less storage space than the existing schemes.

Finally, we provide a proof of concept implementation in Section VI, and conclude in Section VII.

II. Preliminaries

A (k, n)-*threshold secret sharing scheme* [18] allows to distribute a document across n storage servers, such that at least k storage servers must collaborate to reconstruct the document while any subset smaller than k learns nothing about the document shared. More precisely, secret sharing is composed of two algorithms: algorithm Share allows to store a document in distributed fashion by generating and distributing shares to a set of n storage servers $\mathcal{S} = \{S_1, \ldots, S_n\}$, and algorithm Recontruct, where subsets $A \subset \mathcal{S}$ of these storage servers reveal their shares to retrieve the document stored. While in threshold secret sharing any k-subset of storage servers can reconstruct the document, i.e. if $|A| \geq k$, this is different for *weighted* and *hierarchical* secret sharing.

Using *weighted secret sharing* [18] each storage server S_i for $i \in 1, \ldots, n$ is accompanied with a weight w_i determining how many shares of the document it receives. A subset of storage servers A can retrieve a document if the sum of their weights is larger than or equal to k, i.e. if $\sum_{S_i \in A} w_i \geq k$. It follows that the more shares a storage server has the higher is its reconstruction power, because the less shares it needs from other storage servers to reconstruct the document.

In *hierarchical secret sharing*, e.g. see [19], each storage server S_i, for $i \in 1, \ldots, n$, is accompanied with a weight w_i that assigns it to a level in a hierarchy. More precisely, a hierarchy with a total number of $\ell \leq n$ levels is spanned, i.e. $\mathcal{L} = \{L_1, \ldots, L_\ell\}$, where L_1 is the highest level and L_ℓ the lowest. In addition, a threshold k_h is assigned to each level L_h, for $h \in 1, \ldots, \ell$, such that $0 < k_1 < \cdots < k_\ell$. Under which condition a subset $A \subset \mathcal{S}$ of storage servers can retrieve the document depends on the hierarchical secret sharing scheme used. In *disjunctive secret sharing* [20] the document is retrieved if **at least one level** L_h, with $h \in 1, \ldots, \ell$, can be found, such that subset $A \subset \mathcal{S}$ contains at least k_h storage servers assigned to levels equal or higher than L_h. In *conjunctive secret sharing* [20] the document is retrieved if **for all levels** L_h, for $h := 1, \ldots, \ell$, subset $A \subset \mathcal{S}$ contains at least k_h storage servers that are assigned to a level equal or higher than L_h. In both cases the higher the level of a storage server is, the higher its reconstruction power is. A storage server that is assigned to level L_1, for instance, can replace any storage server on any level when conjunctive secret sharing is used. In disjunctive secret sharing a certain amount of storage servers assigned to level L_1 is even required to form a subset that is able to reconstruct the document.

Social secret sharing schemes use either weighted [11], [13], [12] or hierarchical [14] secret sharing, but determine the weights of the storage servers depending on their behavior. In addition, on a regular basis these weights are updated and the shares are reshared. More precisely, a social secret sharing scheme is defined as the tuple (Share, Tune, Recontruct), where algorithm Share allows to store a document in distributed fashion and algorithm Recontruct to retrieve it. The additional algorithm Tune is used to determine and update the weights of each storage server and adjust accordingly the shares distributed.

Dynamic secret sharing [21] is defined as the tuple (Share, Add, Reset, Recontruct). The algorithms Share and Recontruct allow to store and retrieve a document, respectively. The algorithms Add and Reset are interactive protocols between the storage servers. More precisely, algorithm Add allows to generate additional shares to a shared document and therefore to enlarge the set of storage servers. Note that this algorithm does not change the shares held by the "old" storage servers. Algorithm Reset renews all shares and allows to change the parameters of the secret sharing scheme, e.g. the threshold, and to add, remove, and replace storage servers. Note that all these changes can be performed without revealing any information about the document shared, nor involving the document owner in this process.

Solutions that provide these algorithms are available for both weighted secret sharing, e.g. [8], [5], and hierarchical secret sharing [21]. In weighted secret sharing the parameters changed when calling algorithm Reset are the total number of storage servers n and the threshold k. A dynamic hierarchical secret sharing scheme allows to change the total number of storage servers n, the total number of levels ℓ, and all thresholds $k_1, \ldots, k_\ell$ assigned to the individual levels.

III. Adaptive Social Secret Sharing

In this section, we define our framework for adaptive social secret sharing, AS^3, including parameters and algorithms. Furthermore, we provide the set of rules these parameters must satisfy in order for the scheme to be well defined at any time. Afterwards, we present all algorithms in detail.

A. Framework Overview

In social secret sharing a user wants to store a document in distributed fashion. More precisely, it selects a set of storage servers coming from different cloud providers, such as Google, Amazon, or Microsoft, on which it wants to store shares of its document. In addition, the behavior of the selected storage servers is monitored such that the reconstruction power of each storage server can be adapted accordingly. Furthermore, the secret sharing scheme allows the user to add and remove storage servers at any time.

For our framework, we assume that the social secret sharing scheme is run over a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n storage servers owned by different cloud providers. Then, *adaptive social secret sharing* is defined as the tuple (Share, Tune, Reset, Recontruct). Algorithm Share allows a user to store its document using either a dynamic weighted [11], [12] or a dynamic hierarchical secret sharing scheme [21]. In both cases the initial weights $w_1, \ldots, w_n$ for the storage servers $S_1, \ldots, S_n$ are determined and algorithm S.Share of the underlying secret sharing scheme is called with the vector of weights as input. Afterwards, the storage servers run periodically algorithm Tune to determine and update the weights of each storage server and to accordingly adjust the shares by calling the algorithm S.Reset. Then, algorithm Reset allows the user to add, remove, and replace storage servers by first computing weights for the new comers and then calling algorithm S.Reset of the underlying secret sharing scheme with the new storage servers and weights as input. Finally, at any point in time the user can retrieve the document by running

algorithm Reconstruct, which calls the specific algorithm S.Reconstruct of the underlying secret sharing scheme.

The basic difference between the adaptive social secret sharing scheme presented here and the existing social secret sharing schemes are the algorithms Tune and Reset. The algorithm Tune we propose differs from the original proposal by Nojoumian and Stinson [11], [12] because the storage servers are monitored by a trust system that takes into account two behaviors rather than one (see Section IV). Specifically, the behavior of the storage servers is monitored with respect to both confidentiality and retrievability, and not with respect to retrievability only. In addition, the algorithm Tune is responsible not only for adjusting the shares according to the new trust values, but also for adjusting the threshold and checking if the new parameters lead or not to instable states where the document can be lost or its confidentiality is violated. The existing approaches can adjust the quantity of the shares or the level they are assigned to, but they cannot adjust the threshold. Thus, they lead often to instable states and inefficient solutions (see Section V). Instead, we propose a more complex algorithm that uses the algorithm S.Reset of the underlying dynamic weighted or dynamic hierarchical secret sharing scheme. This allows to adapt the parameters, such that the shares distribution leads to an optimal storage space consumption and computation overhead and does not put the confidentiality nor the retrievability of the document in danger. In addition, the user can call algorithm Reset to add, remove, or replace storage servers. This algorithm uses the bootstrapping mechanism presented in Section IV, which allows to compute adequate weights for newcomer storage servers.

Our scheme is parameterized by the following values. (1) The total number of storage servers n which is input to algorithm Share and can be changed by calling algorithm Reset. (2) The weights $w_1, \ldots, w_n$ of storage servers $S_1, \ldots, S_n$, respectively, which are initialized by algorithm Share and updated regularly by algorithm Tune. If we use weighted secret sharing as underlying secret sharing scheme we also use (3a) threshold k required to reconstruct the document while for hierarchical secret sharing we have (3b.1) the total number ℓ of levels and (3b.2) thresholds $k_1, \ldots, k_\ell$ for levels $L_1, \ldots, L_\ell$, respectively. In addition, m denotes the total number of different subsets of storage servers that are able to reconstruct the document.

There are two aspects that distributed storage systems should guarantee and each of them deals with a specific behavior of the storage servers. On the one hand, there is the confidentiality of the document and this concerns the will of the storage servers to collude to get information about the document or to reveal the shares to a third party (e.g. governmental agencies). These storage servers run the programs and algorithms correctly and are referred to as *honest but curious*. On the other hand, there is the retrievability of the document and this concerns the availability and response time of the storage servers when algorithms Tune, Reset, and Recontruct are run and the validity of the shares they submit. These storage servers might be under the control of an attacker and are referred to as *faulty*. The parameters defined above have to be set, such that the document is kept secret and can be retrieved. In order to do that, these parameters have to satisfy the following two rules.

Rule 1. This rule addresses the confidentiality of the document. The smallest subset of storage servers that is able to run algorithm Reconstruct and to retrieve the document must be strictly greater than the number of storage servers that is assumed to be honest but curious.

Rule 2. This rule addresses the retrievability of the document. A certain threshold of storage servers is needed to perform algorithms S.Reset and S.Recontruct of the underlying dynamic secret sharing scheme and correspondingly to run algorithms Tune, Reset, and Recontruct of the adaptive social secret sharing scheme. Thus, the parameters must be chosen such that the system can cope with faulty storage servers.

We now analyze under which conditions these rules are fulfilled for the different secret sharing schemes. Note that these conditions are defined taking into account the worst case scenario both from confidentiality and retrievability point of view.

For Rule 1, the worst case is when the x honest but curious storage servers are those with the highest reconstruction power. Thus, assume for weighted (hierarchical) secret sharing that W is defined as the vector of length n containing all the weights (levels) $w_1, \ldots, w_n$ sorted from the smallest value (lowest level) to the highest value (highest level), i.e. $W := \{W[1], \ldots, W[n]\}$ where $W[i] \leq W[i+1]$, for $i = 1, \ldots, n$. Thus, for instance, $W[n-x+1]$ denotes the weight (level) the least powerful of the x honest but curious storage servers is assigned to. For *weighted secret sharing*, Rule 1 is fulfilled if $k > \sum_{i \geq n-x+1} W[i]$. For *hierarchical secret sharing*, two further cases are distinguished. Let us denote by k_h the threshold of level L_h and by $x_h \leq x$ the number of honest but curious storage servers assigned to this level, for $h = 1, \ldots, \ell$. For *disjunctive secret sharing*, Rule 1 is fulfilled if $\forall\, h = 1, \ldots, \ell \;\; k_h > \sum_{i \leq h} x_i$. For *conjunctive secret sharing*, Rule 1 is fulfilled if $\exists\, h$, such that $k_h > \sum_{i \leq h} x_i$.

For Rule 2, the worst case is when the x' faulty storage servers are those with the highest reconstruction power. In this case, it must be ensured that there is still at least one subset of non faulty storage servers able to retrieve the document, i.e. $m \geq 1$. We denote by $\mathcal{S}' \subset \mathcal{S}$ the subset of all storage servers where the x' most powerful storage servers are discarded (in this case, the ones corresponding to the last x' entries of vector W). For *weighted secret sharing*, m is defined as $m := \left\{ A \subset \mathcal{S}' | \sum_{S_i \in A} w_i \geq k \right\}$ and trivially Rule 2 is fulfilled if $m \geq 1$. For *hierarchical secret sharing*, we additionally assume that $n'_h \leq n_h$ is the amount of storage servers from $\mathcal{S}'$ assigned to level L_h, for $h = 1, \ldots, \ell$. For *disjunctive secret sharing*, Rule 2 is fulfilled if $\exists\, h$ such that $\sum_{i \leq h} n'_i \geq k_h$. For *conjunctive secret sharing*, Rule 2 is fulfilled if $\forall\, h = 1, \ldots, \ell$ it holds that $\sum_{i \geq h} n'_h \geq k_h$. Notice that Rule 2 is harder to fulfill using conjunctive secret sharing, since for all levels at least k_h storage servers are needed that are assigned to a level equal or higher than L_h. Thus, the chance that a failure leads to instable states where the document cannot be retrieved any more is significantly higher. Since this makes *disjunctive secret sharing* the most suitable hierarchical secret sharing scheme to be used in social secret sharing we do not consider *conjunctive secret sharing* any further.

B. Share

When algorithm Share is run, the parameters for the social secret sharing scheme are initialized. First, a set of storage servers is selected and the parameters n and k for weighted secret sharing and n, ℓ, and $k_1, \ldots, k_\ell$ for hierarchical secret sharing are set, such that Rule 1 and Rule 2 are fulfilled. More precisely, it is common practice to choose x, x', and n, such that $n = 3x' + 1$ and $x = x'$ [4]. In fact, this further constraint on n ensures the correctness of the shares[1]. Then threshold k (for the weighted secret sharing) the thresholds $k_1, \ldots, k_\ell$ (for the disjunctive secret sharing) are chosen such that Rule 1 is fulfilled. Note that in fact the threshold of the underlying secret sharing scheme ensures the confidentiality of the document as it prevents non authorized subsets from getting any information about it. More precisely, for weighted secret sharing, Rule 1 is fulfilled by setting k to $x + 1$. For disjunctive secret sharing, Rule 1 is fulfilled by setting k_1 to $x + 1$ and k_{h+1} to $k_h + 1$, for $h = 1, \ldots, \ell - 1$. Afterwards, the weights $w_1, \ldots, w_n$ are selected. More precisely, the same weight w_i is assigned to each storage server S_i, for $i = 1, \ldots, n$. For weighted secret sharing, each storage server has weight $w_i = 1$. For disjunctive secret sharing, a single level L_1 is spanned and each storage server is assigned to this level. For example, algorithm Share(x, x') called with input $x = x' = 3$ sets up $n = 10$ storage servers $S_1, \ldots, S_{10}$. Then, for weighted secret sharing, it chooses threshold $k = 4$ and weight $w_i = 1$ for each storage server. For disjunctive secret sharing, it spans one level L_1 with threshold $k_1 = 4$ and assigns each storage server to this level.

C. Tune

Algorithm Tune is responsible for adjusting the weights of the storage servers according to their behaviors, both with respect to confidentiality and availability. To do this, algorithm Tune calls the trust system and requests for each storage server the measured behavior with respect to confidentiality c_i and retrievability r_i, for $i = 1, \ldots, n$ (for details see Section IV). Based on these values, first, a (new) estimation of x and x' is computed, i.e. the expected amount of honest but curious and faulty storage servers are determined. Second, the distribution of the shares and the parameters of the social secret sharing scheme are adjusted, such that the document is shared according to the measured behavior and Rule 1 and Rule 2 are fulfilled. In the following, details about both steps are provided.

1) Estimation of x and x': The estimation of x and x' is performed as follows: two thresholds $t_c, t_r \in [0, 1]$ are defined for confidentiality and retrievability, respectively. If the trust value of a storage server with respect to confidentiality is below t_c, then this storage server is considered honest but curious. The parameter x counts how many of such storage servers are in the system. Faulty storage servers are similarly individuated using threshold t_r as discriminant value and counted by parameter x'. Then, it is checked that there are still enough honest storage servers to ensure confidentiality and enough robust (i.e. non faulty) storage servers to ensure retrievability, i.e. $x < n$ and $x' < n$. These two checks are necessary yet not sufficient conditions such that, respectively, Rule 1 and Rule 2 are fulfilled. Note that during the instantiation of the parameters (Section III-B), n and k are chosen, such that these conditions are fulfilled. Afterwards, x and x' can increase or decrease even though the total number of storage servers n does not change violating Rule 1 or Rule 2. Thus, each time algorithm Tune is run these two checks have to be performed. If these two constraints are not satisfied, then a failure message is transmitted and the user is asked to call algorithm Reset to add additional storage servers and/or to remove the untrustworthy ones.

2) Adjusting Share distribution: If $x < n$ and $x' < n$ hold, then algorithm Tune adjusts the shares and parameters, such that the document is shared according to the measured reputation and Rule 1 and Rule 2 are fulfilled. More precisely, for weighted secret sharing new weights $w_1, \ldots, w_n$ are computed and, accordingly, a new threshold k is set. For disjunctive secret sharing, new weights $w_1, \ldots, w_n$ are computed and, accordingly, ℓ levels $L_1, \ldots, L_\ell$ are spanned together with their respective thresholds $k_1, \ldots, k_\ell$.

First, the weights are computed taking into account the two behaviors confidentiality and retrievability. More precisely, let us denote by $c_i \in [0, 1]$ and $r_i \in [0, 1]$ the outputs of the trust system concerning the behavior of storage server S_i for confidentiality and retrievability, respectively. In addition, let the integer $\beta > 0$ denote the accuracy with which the trust values should be mapped to the shares distribution. Note that β is a parameter chosen by the user according not only to the specific document shared, but also to the storage space and the computational overhead it wants to consume (and pay for). In fact, the higher the value of β, the more shares are generated and need to be stored. Ratio u_i determines how much reconstruction power storage server S_i receives. It is computed as a convex combination of c_i and r_i. Specifically, two real numbers $\lambda_c, \lambda_r \in [0, 1]$, the first one for confidentiality and the second one for retrievability, are chosen such that $\lambda_c + \lambda_r = 1$ and ratio $u_i \in [0, 1]$ is computed as $u_i := \lambda_c \cdot c_i + \lambda_r \cdot r_i$, for $i = 1, \ldots, n$. Then, interval $[0, 1]$ is divided into β disjoint subintervals, $I_1 = [0, \frac{1}{\beta}), I_2 = [\frac{1}{\beta}, \frac{2}{\beta}), \ldots, I_\beta = [1 - \frac{1}{\beta}, 1]$. For weighted secret sharing, storage server S_i whose ratio u_i lies in interval I_j receives j shares, for $i = 1, \ldots, n$. For disjunctive secret sharing, β corresponds to the amount ℓ of levels $L_1, \ldots, L_\ell$ spanned and storage server S_i whose ratio u_i lies in I_j is assigned to level L_j, for $i = 1, \ldots, n$ (as initially proposed by [14]).

Second, thresholds k or $k_1, \ldots, k_\ell$ are chosen such that Rule 1 is fulfilled. The validity of Rule 1 is checked as discussed in Section III-A. For the weighted secret sharing, k is set to $1 + \sum_{i>n-x} W[i]$. For disjunctive secret sharing, k_1 is set to $x + 1$ and k_{h+1} to $k_h + 1$, for $h = 1, \ldots, \ell - 1$. Note that k and $k_1, \ldots, k_\ell$ are set to the minimum value such that Rule 1 is satisfied. Greater values are still valid, but they increase the computational overhead and the chance that too many storage servers fail to retrieve the document.

Third, it is checked weather the shares distribution leads to m sufficiently many subsets that can reconstruct the document. That is, the validity of Rule 2 is checked as discussed in Section III-A. If Rule 2 is not satisfied, then the new shares are not computed and distributed, because otherwise the document would be irreversibly lost. Instead, a warning message is sent

[1]When dealing with a bounded adversary that can corrupt up to x' storage servers over time, then the constraint $n \geq 3x' + 1$ is sufficient for correctness. In case the adversary is not bounded, then correctness is achieved using commitments schemes (see [3], [15]).

to the user to make it aware of the possibly dangerous situation caused by the presence of too many faulty storage servers. The user is strongly recommended to reboot new storage servers to prevent the loss of the document. Note that in this paper we require for simplicity that $m \geq 1$, as it is the lower bound such that Rule 2 is satisfied. However, depending on the application, more robustness in terms of reconstruction might be preferred and a higher value for m might be required.

Finally, algorithm $\mathsf{S.Reset}$ of the underlying weighted or hierarchical secret sharing scheme is called. It adapts the parameters of the secret sharing scheme and the weights assigned to each storage server according to the values computed by algorithm Tune. For example, let us assume the reputation system outputs $(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}) = (0.40, 0.32, 0.5, 0.20, 0.35, 0.58, 0.87, 0.10, 0.25, 0.92)$ and $(r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}) = (0.15, 0.28, 0.40, 0.60, 0.37, 0.18, 0.53, 0.21, 0.80, 0.44)$. Let us further assume algorithm $\mathsf{Tune}(\mathsf{t_c}, \mathsf{t_r}, \lambda_\mathsf{c}, \lambda_\mathsf{r}, \beta,)$ is called with $(0.3, 0.2, 0.5, 0.5, 3)$. Then, it estimates $x = 3$ honest but curious storage servers and $x' = 2$ faulty storage servers and computes ratios $(u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}) = (0.27, 0.30, 0.45, 0.4, 0.36, 0.38, 0.70, 0.15, 0.52, 0.68)$. For weighted secret sharing, it assigns weight 1 to storage servers S_1, S_2, S_8, weight 2 to S_3, S_4, S_5, S_6, S_9, weight 3 to S_7, S_{10}, and sets threshold $k = 8$. For disjunctive secret sharing, it spans three levels L_1, L_2, L_3 with thresholds $k_1 = 4, k_2 = 5, k_3 = 6$, respectively. It assigns S_7, S_{10} to level L_1, S_3, S_4, S_5, S_6, S_9 to level L_2, and S_1, S_2, S_8 to level L_3. In both cases $m \geq 1$, so no warning message is sent.

D. Reset

Algorithm Reset takes as input threshold k or thresholds $k_1, \ldots, k_\ell$ adjusted by algorithm Tune, computes and distributes the shares to the storage servers in the storage system according to weights $w_1, \ldots, w_n$, and, eventually, the shares of newcomers. Regarding weighted secret sharing scheme, if the threshold k has been modified, then algorithm $\mathsf{S.Reset}$ described in [5] is run and the shares are distributed according to $w_1, \ldots, w_n$. Regarding hierarchical secret sharing scheme, if thresholds $k_1, \ldots, k_\ell$ have been modified, then algorithm $\mathsf{S.Reset}$ introduced in [21] is run and the shares are distributed according to weights $w_1, \ldots, w_n$. For example, for weighted secret sharing, algorithm $\mathsf{Reset}(\mathsf{k}, \mathsf{w_1}, \mathsf{w_2}, \mathsf{w_3}, \mathsf{w_4}, \mathsf{w_5}, \mathsf{w_6}, \mathsf{w_7}, \mathsf{w_8}, \mathsf{w_9}, \mathsf{w_{10}})$ called with $(8, 1, 1, 2, 2, 2, 2, 3, 1, 2, 3)$ sets a polynomial of degree seven to share the secret and distributes one share to storage servers S_1, S_2, S_8, two shares to storages servers S_3, S_4, S_5, S_6, S_9, and three shares to storage servers S_7, S_{10}. For disjunctive secret sharing, algorithm $\mathsf{Reset}(\mathsf{k_1}, \mathsf{k_2}, \mathsf{k_3}, \mathsf{L_1}, \mathsf{L_2}, \mathsf{L_3}) = \mathsf{Reset}(4, 5, 6, \{\mathsf{S_7}, \mathsf{S_8}\}, \{\mathsf{S_3}, \mathsf{S_4}, \mathsf{S_5}, \mathsf{S_6}, \mathsf{S_9}\}, \{\mathsf{S_1}, \mathsf{S_2}, \mathsf{S_8}\})$ sets a polynomial of degree five to share the secret and distributes shares from such polynomial where S_1, S_2, S_8 are assigned to level L_3, S_3, S_4, S_5, S_6, S_9 are assigned to level L_2, and S_7, S_{10} are assigned to level L_1.

E. Reconstruct

Algorithm $\mathsf{Reconstruct}$ is called when the user wants to retrieve the document. Depending on the underlying secret sharing scheme, a specific algorithm $\mathsf{S.Reconstruct}$ is run. Specifically, algorithm $\mathsf{S.Reconstruct}$ described in [12] or [18] is called for weighted secret sharing, while algorithm $\mathsf{S.Reconstruct}$ described in [14] or [21] is called for hierarchical secret sharing.

IV. Trust System for Social Secret Sharing Scheme

One basic ingredient of our adaptive social secret sharing scheme is to determine the trustworthiness of each of the storage servers based on their behavior. In this paper, we are interested in the behavior of the storage servers regarding *confidentiality* and *retrievability*. More precisely, we use a trust system to compute the trust values of the storage servers needed by our Tune algorithm (see Section III). The trust values are based on direct and indirect experiences (or evidence), either obtained via direct interactions or via witness referrals [10]. The procedure we follow is composed of three steps. First, evidence collection and processing, second, trust computation, and third, trust bootstrapping.

A. Evidence Collection and Processing

Evidence with respect to confidentiality and retrievability of storage servers can be collected from several parties. In the context of distributed storage, the following parties are involved: storage servers (n) and a user (u) that uses the social secret sharing scheme to store its document. In this paper, we refer to the participants providing evidence as *submitters*. A piece of *evidence* is a binary value assigned to a storage server by a submitter: 1 indicates that the submitter believes the storage server behaves in a *good* manner with respect to a certain behavior (i.e. either confidentiality or retrievability) and 0 indicates that the submitter believes the storage server behaves in a *bad* manner.

For confidentiality, "good behavior" means that the storage server follows the protocol and does not reveal the information stored to a third party (e.g. a governmental agency), nor does it conspire with other storage servers to retrieve or gain knowledge about the document. Vice versa, "bad behavior" means that the storage server follows the protocol, but reveals the information stored to a third party, or it conspires with other storage servers to retrieve or gain knowledge about the document. In this case, bad behaving storage servers are referred to as *honest but curious*.

For retrievability, "good behavior" means that the storage server responds immediately with the correct shares as soon as $\mathsf{Reconstruct}$ algorithm is called by the user (u) to retrieve the document. Vice versa, "bad behavior" means that the storage server does not respond, responds later than expected, or provides inconsistent shares when the user wants to retrieve the document. In this case, bad behaving storage servers are assumed to be either controlled by an attacker or be broken. We refer to those as *faulty* storage servers.

After the reputation system collects all pieces of evidence from the submitters, the evidence is processed to determine the *trust value* of a particular storage server. The processing can be performed according to well-established trust models, CertainTrust [16] or Subjective Logic [9]. In this paper, we use CertainTrust although Subjective Logic can be readily substituted, as both models are isomorphic. In CertainTrust,

trust values are represented as tuples $o = (t, c)$, where t is the ratio between the amount of positive (good) evidences received from the submitters over the sum of the total amount of evidences (good and bad) collected. The second value c is the certainty level associated with the collected evidences. Denoted by e_p the amount of positive evidences and by e_n the amount of negative evidences, the certainty level c is computed as $c = \frac{N \cdot (e_p + e_n)}{2 \cdot (N - e_p - e_n) + N \cdot (e_p + e_n)}$.

The parameter N refers to the maximum number of evidence required to reach the highest certainty level and usually depends on specific application scenarios. Let us assume that a set of submitters consists of a set of n storage servers and a user (u), which stored its document using the social secret sharing scheme. In this scenario, we expect for each protection goal, confidentiality and retrievability, to retrieve evidence from all storage servers (s) except the evaluated one, i.e. $N_s = n - 1$, and one evidence from the user (u), i.e. $N_u = 1$. Then, two trust values for confidentiality, i.e. $o_{C,s}$ and $o_{C,u}$, and two trust values for retrievability, i.e. $o_{R,s}$ and $o_{R,u}$, are computed from the evidences received from the storage servers and the user respectively.

B. Trust Computation

The outcome of the evidence collection and processing phase are a set of trust values for confidentiality and a set of trust values for retrievability from different groups of submitters. In our example the two groups are storage servers and users. An additional input to this process is a vector of weights that indicates how much the retriever of the trust values, i.e. the social secret sharing scheme on behalf of the user, relies on the collected evidences. A user might prefer the trust value based on its collected evidence than based on the evidence submitted by the storage servers. Thus, the weight for the storage servers $\omega_s \in [0, 1]$ is smaller than the weight for the user $\omega_u \in [0, 1]$, i.e. $\omega_s < \omega_u$. These weights can either be provided by the users or be public parameters.

Next, the trust value c_i for a storage server S_i with respect to confidentiality is computed as $c_i = o_{C_s} \bigoplus_{\omega} o_{C_u}$, where $\bigoplus_{\omega}$ refers to the *weighted fusion* with respect to the weights ω_s and ω_u (see [6]). Similarly, the trust value r_i for storage server S_i with respect to confidentiality is computed as $r_i = o_{R_s} \bigoplus_{\omega} o_{R_u}$. This process is repeated for all storage servers and the complete vector of trust values for all storage servers with respect to confidentiality $c_1, \ldots, c_n$ and with respect to retrievability $r_1, \ldots, r_n$ are provided to the social secret sharing scheme (see Section III).

C. Trust Bootstrapping

In the case of newcomer storage servers, evidence regarding their behavior is unknown. Thus, the trust system requires a bootstrapping process in order to initialize their trust values. A growing number of cloud providers are publishing their service-specific security capabilities in a public repository, e.g. the CSA STAR [2] since 2011. Computational trust methods [7] are leveraged to quantify the level of security capabilities (CSA STAR) of cloud services by means of trust values. These trust values can be used to determine initial trust values for newcomer storage servers.

In the CSA STAR, cloud providers publish one set of valid answers in response to a questionnaire, i.e. Consensus Assessment Initiative Questionnaire (CAIQ) regarding each of the service they offer. The answers are considered as evidence of existing security capabilities that cloud providers claim to have for their services. The CAIQ (v1.1) has 11 security domains which are as follows: Compliance (CO), Data Governance (DG), Facility Security (FS), Human Resources Security (HR), Information Security (IS), Legal (LG), Operations Management (OP), Risk Management (RI), Release Management (RM), Resiliency (RS), and Security Architecture (SA).

Trust values (t, c) are calculated using the CertainTrust model for each of the mentioned CAIQ domain based on the evidence (assertions) provided by the cloud providers in the CSA STAR. In order to compute the trust value of a cloud service, the CertainLogic AND ($\wedge$) [17] operator is used to combine all the trust values (t, c) calculated for each of the security domains. The overall trust value (t, c) is the initial value for the newcomer storage server that has joined the pool of storage servers for the first time. Once the evidence regarding their behavior are available, the trust system replaces the initial trust values with the trust values computed based on the storage server's behavior.

V. Related Work and Comparison with AS^3

In this section, we provide an evaluation of the related work with respect to social secret sharing, i.e. the weighted social secret sharing by Nojoumian and Stinson [13], [12] and the hierarchical social secret sharing by Pakniat et al. [14]. We briefly describe the approaches and highlight their shortcomings. Furthermore, we show the countermeasures provided by our adaptive weighted and adaptive hierarchical social secret sharing scheme AS^3 presented in Section III.

A. Weighted Social Secret Sharing

Algorithm Share sets the trust values $t_1, \ldots, t_n$ to zero and the weights $w_1, \ldots, w_n$ according to a specific distribution. Then, shares are generated and distributed to the storage servers according to their weights. Algorithm Tune calls the trust function which updates the trust values $t_1, \ldots, t_n$ based on the response time of the storage servers $S_1, \ldots, S_n$. Then, algorithm Tune adjusts the weights $w_1, \ldots, w_n$ as follows. The storage servers whose weight increased get one additional share while one share is taken from those whose weight decreased. Furthermore, each w_i is bounded by a parameter z much smaller than the threshold k of the scheme.

However, this approach has left many issues unsolved.

(1) Initialization of weights and threshold: It is not specified how the weights $w_1, \ldots, w_n$ are initialized because no detail is given regarding which distribution to choose. Furthermore, it is not specified how to select threshold k given the weights. Note that care must be taken when this parameter is selected, because it is of major importance to ensure both confidentiality and retrievability. In fact, on the one hand, threshold k determines the maximum number $(k-1)$ of colluding storage servers the scheme can cope with. On the other hand, it determines how many shares held by different storage servers are needed to reconstruct the document.

(2) Translation of trust values into weights: It is not defined how the trust values $t_1, \ldots, t_n$ affect the weights $w_1, \ldots, w_n$. More precisely, it is not defined if increase or decrease of weight w_i happens because the trust value t_i is, respectively, above or below a certain threshold or because the trust value t_i itself is, respectively, greater or smaller than the trust value of the previous round. In the former case, it is left open how to choose these thresholds. In the latter case, storage servers that behave better compared to the last round would be rewarded even though they have a low trust value and can therefore be considered untrustworthy. Furthermore, it is not clear which ranges of different trust values are mapped to the same weight. Note that having small ranges, e.g. such that each single trust value is mapped to one weight, leads to a huge amount of shares, causing a high computational overhead and storage space consumption.

(3) Trust in confidentiality: The trust values $t_1, \ldots, t_n$ are computed taking into account only the behavior of the storage servers with respect to retrievability, while the confidentiality aspect is not addressed.

(4) Selection of upper bound z: No indication is given with respect to the choice of the upper bound z for the weights $w_1, \ldots, w_n$. The parameter z is introduced to prevent the storage servers from having a weight high enough to reconstruct the document by themselves, violating confidentiality. However, the parameter z must be chosen such that threshold k is much larger than any single initial weight w_i. Thus, there is no guarantee that all the weights of all the storage servers together can actually reach threshold k, namely there is no guarantee that the document can be reconstructed. Another problem is that parameter z makes the trust system less effective: the weights $w_1, \ldots, w_n$ cannot go beyond z no matter how high the trust values $t_1, \ldots, t_n$ are. Thus, approaching z, it is less and less rewarding for the storage servers to behave well. Note that, choosing such a high threshold k leads to a high computation overhead, which is not necessary when the weights are small.

(5) Instable states: Wrong choices for threshold k and bound z might lead to "instable states". In this framework, we refer to an instable state as a state in which either the document cannot be reconstructed anymore or colluding storage servers are able to reconstruct the document. The current solution does not provide any measures to prevent or detect such states.

(6) Dynamism of parameters: Threshold k and bound z are selected by algorithm Share and kept unchanged for the entire lifetime of the storage. This has several drawbacks. If, for instance, the amount of storage servers and/or the amount of shares generated got increased, then also threshold k and bound z must be increased as well to prevent malicious storage servers from reconstructing the document.

Our weighted AS^3 scheme addresses the issues summarized above. With respect to (1), algorithm Share specifies how to initialize the weights $w_1, \ldots, w_n$ and threshold k. Furthermore, algorithm Tune defines how to map the trust values into weights thereby addressing issue (2). More precisely, our trust system outputs two trust values: one to measure the behavior of the storage servers with respect to confidentiality and one with respect to retrievability (see Section IV). Thus, we solve the shortcoming described in (3). With respect to (4) our scheme employs an accuracy parameter β that keeps bounded the total amount of shares generated, instead of bounding the weights. This approach optimizes the storage space consumption and leads to a low computation overhead. Furthermore, the fact that there is no upper bound z encourages the storage servers to always behave well. A very critical part of all social secret sharing schemes is reaching instable states as described in shortcoming (5). We address this by providing two rules: Rule 1 protects confidentiality and Rule 2 protects retrievability. Note that the compliance with these rules is checked *before* new shares are generated and distributed. This prevents that the document is revealed or lost. In addition, the scheme takes measures to prevent instable states. For instance, algorithm Share sets the total amount of storage servers n such that the Byzantine model [4] is satisfied, i.e. such that with the initial estimation of untrustworthy storage servers the document can be retrieved. Finally, our scheme is the first that provides dynamism with respect to the parameters selected, i.e. n, k, β. In fact, algorithm Tune increases or decreases threshold k at each point in time in order to protect confidentiality (Rule 1) and to ensure retrievability (Rule 2). This especially allows to choose threshold k always tight thereby optimizing the computational overhead and the storage space consumption.

B. Hierarchical Social Secret Sharing

Hierarchical social secret sharing has been introduced by Pakniat et al. in [14]. The underlying scheme is the disjunctive secret sharing scheme. Algorithm Share initializes the trust values $t_1, \ldots, t_n$, the weights $w_1, \ldots, w_n$, and the levels $L_1, \ldots, L_\ell$. The total number ℓ of levels determines the accuracy for which the trust values are mapped into levels. Depending on the weights, the storage servers are assigned to a specific level and shares are generated and distributed accordingly. Algorithm Tune calls the trust function which updates the trust values $t_1, \ldots, t_n$ based on the behavior of the storage servers $S_1, \ldots, S_n$. Then algorithm Tune adjusts the weights $w_1, \ldots, w_n$ and reassign the storage servers to the corresponding levels.

In this approach, the accuracy with which trust values are mapped into levels is provided. However, this does not fully address (4), because the total number of levels ℓ also bounds how much storage servers can be rewarded or punished. In fact, storage servers assigned to level L_1 cannot be rewarded any further and storage servers assigned to level L_ℓ cannot be punished any further, no matter how, respectively, good or bad they behave. Besides this partial countermeasure to (4), this hierarchical social secret sharing scheme leaves the same issues open as identified for the weighted social secret sharing scheme. More precisely, it is not clear how to initialize the weights $w_1, \ldots, w_n$, how many levels $L_1, \ldots, L_\ell$ to span, and how to choose the corresponding thresholds $k_1, \ldots, k_\ell$ (1). Furthermore, it is not defined how the trust values are mapped into weights (2), only trust in reliability is addressed (3), instable states are not detected nor prevented (5), and the parameter selection is not dynamic (6). Our dynamic AS^3 scheme addresses these shortcomings similarly as described for weighted secret sharing.

VI. Implementation

In addition to the schemes we also provide a proof of concept implementation. The user is allowed to adjust certain parameters depending on the type of document and the financial effort it wants to put into the storage. For instance, the higher the number n of storage servers the more expensive the storage. If the user stores sensitive personal health data, then confidentiality is a more critical aspect than retrievability and λ_c and t_c can be set larger than λ_r and t_r, respectively. For other medical data, such as the blood group of a patient, fast retrieval is more critical and λ_r and t_r can be set larger than λ_c and t_c. Furthermore, when collecting experiences there are different groups of submitters that provide opinions, e.g. document owners and storage servers. In this case the user can choose the weights ω_s and ω_u which determine how much to trust in the different groups of submitters. Furthermore, an additional parameter $\varepsilon > 0$ can be introduced to allow the user to decide how much the trust values of the individual storage servers must change to run the adjusting share distribution subprocedure of algorithm Tune. This prevents that shares are updated although the trustworthiness of the storage servers did not change.

We run the weighted social secret sharing (WSSS) [11], [12], the hierarchical social secret sharing (HSSS) [14], and our weighted (W-AS^3) and hierarchical (H-AS^3) adaptive social secret sharing with input parameters $\lambda_c = \lambda_r = 0.5$, $t_c = t_r = 0.2$ for $n = 7, 12, 18$ and $\beta = 2, 3, 4$, respectively. We run 1000 iterations and collected at which round Rule 1 and Rule 2 were on average not satisfied (see Table I). For WSSS and HSSS, Rule 1 fails as soon as less than the threshold number of storage servers is able to reconstruct the secret, which happened soon. Since already after a few rounds the algorithms abort we did not observe a violation of Rule 2 in our experiments. With respect to W-AS^3 and H-AS^3, Rule 1 never fails by construction. Rule 2 is usually violated when x and/or x' change such that the Byzantine model is not fulfilled anymore. However, our experiments showed that this happens in the average only after Round 21.46 and Round 23.86 respectively.

TABLE I: Violation of Rule 1 and Rule 2 for different schemes.

	WSSS	HSSS	W-AS^3	H-AS^3
Round violating Rule 1	2.8	3.4	> 1000	> 1000
Round violating Rule 2	-	-	21.46	23.86

VII. Conclusion

In this work, we introduced AS^3, a framework for adaptive social secret sharing. Our solution is based on dynamic secret sharing and can be instantiated either with weighted dynamic or with hierarchical dynamic secret sharing. In addition, we showed how trust values to storage servers with respect to confidentiality and retrievability can be computed, how trust values for newcomers can be determined, and how the share distribution can be adapted accordingly. Finally, we implemented and tested our solution. For future work, we plan looking at specific applications, e.g. the storage of medical records, identifying additional requirements, and extending our solution accordingly.

Acknowledgments

This work has been co-funded by the DFG as part of projects "Scalable Trust Infrastructures" and "Long-Term Secure Archiving" within the CRC 1119 CROSSING and the European Union's Horizon 2020 research and innovation program under Grant Agreement No 644962.

References

[1] Ernest F. Brickell. Some ideal secret sharing schemes. In *EUROCRYPT*, pages 468–475, 1989.

[2] CSA. Security, Assurance & Trust Registry (STAR). https://cloudsecurityalliance.org/star/, accessed 28 Jul 2016.

[3] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437, 1987.

[4] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *STOC*, pages 580–589, 2001.

[5] V. H. Gupta and K. Gopinath. G_{its}^2 VSR: an information theoretical secure verifiable secret redistribution protocol for long-term archival storage. In *SISW*, pages 22–33, 2007.

[6] Sheikh Mahbub Habib, Sebastian Ries, Sascha Hauke, and Max Mühlhäuser. Fusion of opinions under uncertainty and conflict - application to trust assessment for cloud marketplaces. In *TrustCom*, pages 109–118, 2012.

[7] Sheikh Mahbub Habib, Florian Volk, Sascha Hauke, and Max Mühlhäuser. Computational trust methods for security quantification in the cloud ecosystem. In *The Cloud Security Ecosystem*, pages 463 – 493. 2015.

[8] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, pages 339–352, 1995.

[9] Audun Jøsang. A logic for uncertain probabilities. *INT J UNCERTAIN FUZZ*, 9(3):279–212, 2001.

[10] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[11] Mehrdad Nojoumian and Douglas R. Stinson. Brief announcement: secret sharing based on the social behaviors of players. In *ACM PODC*, pages 239–240, 2010.

[12] Mehrdad Nojoumian and Douglas R. Stinson. Social secret sharing in cloud computing using a new trust function. In *PST*, pages 161–167, 2012.

[13] Mehrdad Nojoumian, Douglas R. Stinson, and Morgan Grainger. Unconditionally secure social secret sharing scheme. *IET Information Security*, 4(4):202–211, 2010.

[14] Nasrollah Pakniat, Ziba Eslami, and Mehrdad Nojoumian. Ideal social secret sharing using birkhoff interpolation method. *IACR Cryptology ePrint Archive*, 2014:515, 2014.

[15] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[16] Sebastian Ries. Extending bayesian trust models regarding context-dependence and user friendly representation. In *ACM SAC*, pages 1294–1301, 2009.

[17] Sebastian Ries, Sheikh Habib, Max Mhlhuser, and Vijay Varadharajan. Certainlogic: A logic for modeling trust and uncertainty. In *Trust and Trustworthy Computing*, volume 6740, pages 254–261, 2011.

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

[19] Gustavus J. Simmons. How to (really) share a secret. In *CRYPTO*, pages 390–448, 1988.

[20] Tamir Tassa. Hierarchical threshold secret sharing. *J. Cryptology*, 20(2):237–264, 2007.

[21] Giulia Traverso, Denise Demirel, and Johannes Buchmann. Dynamic and verifiable hierarchical secret sharing. In *ICITS*, pages 1–20, 2016.