Social Secret Sharing in Cloud Computing Using a New Trust Function

Mehrdad Nojoumian and Douglas R. Stinson
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
mnojoumi@cs.uwaterloo.ca, dstinson@math.uwaterloo.ca

***Abstract*—We first review the notion of social secret sharing and its trust function. We then illustrate how this construction can be used in cloud computing to create a self-organizing environment. In fact, we show distributed secure systems using threshold secret sharing can be adjusted automatically based on the resource availability of the cloud providers. Accordingly, we propose a new trust function with social characteristics in order to improve the existing social secret sharing scheme.**

Keywords: trust modeling, secret sharing, cloud computing.

I. Introduction

In a *secret sharing* scheme, a secret is divided into n shares in order to be distributed among a set of players. Subsequently, an authorized subset of players collaborate to reconstruct the secret [1] [2]. In particular, a $(t; n)$*-threshold secret sharing* (*TSS*) scheme consists of two phases: *sharing* and *recovery*. All computations are performed in a finite field $\mathbb{Z}_q$ where q is a prime number:

1) **Secret Sharing**: a dealer selects $f(x)$ *2* $\mathbb{Z}_q[x]$ of degree t 1 such that $f(0) =$ is the secret. The dealer then sends shares $f($

To assign multiple shares rather than a single share to some players, *weighted secret sharing* (*WSS*) is introduced in [13]. Suppose in a company, the secret key of the safe deposit box is shared among chief executive officer, director, and two managers. Let assume these parties receive $4; 3; 2; 2$ shares respectively on a polynomial of degree 5. As a result, the chief executive officer can open the safe deposit box with the director or one manager but the other parties can only open it if they all collaborate. Indeed, weighted secret sharing is used to prioritize different players in a hierarchy structure.

A. Motivation and Contribution

We illustrate how *social secret sharing* (*SSS*) can be applied in distributed secure systems using cloud computing infrastructures. Moreover, we intend to improve social secret sharing by proposing a new trust function.

Therefore, as our contributions, we first explain a scenario in which this cryptographic primitive can be used to create a self-organizing protocol in the cloud. In fact, we show a distributed system can be reconfigured automatically based on the resource availability of the cloud providers. Subsequently, we provide a new trust function with social properties in order to improve the existing social secret sharing scheme.

B. Organization

The rest of this paper is organized as follows. Section II reviews the notion of social secret sharing. Section III illustrates a new application of this scheme in cloud computing. Section IV proposes a new trust function. Finally, Section V provides concluding remarks.

II. Review of Social Secret S

for several times in order to gain a high trust value. He can then defect in a critical transaction to severely damage the scheme. By considering this transaction cost parameter, a weight for "cooperation" or "defection" is defined and accordingly the trust value is adjusted.

III. Application in Cloud Computing

In *cloud computing*, different commercial providers (such as Amazon, Google, and Microsoft) offer computing services to consumers. The major goal is to provide "computing", "storage", and "software" as a service. As a result, consumers do not need to invest in IT infrastructure on their own. They can obtain these services from external providers according to their demands by a pay-per-use model [18], i.e., obtaining more services in the case of growing demand and vise versa.

A significant challenge in cloud computing is "resource management" due to the consumers' expectations in terms of resource availability, overall performance, etc. In some settings, enterprises provide valuations to service providers (i.e., the money they are going to pay if cloud providers satisfy their demands). The service providers then try to maximize their own profit, for instance, by prioritizing the consumers' jobs. All these factors may lead to competition, negotiation, dynamic allocation, and automatic load balancing. For an extensive survey on this matter, see [19].

We demonstrate a new method of share distribution over the cloud in a secure system using threshold secret sharing. The question is how such systems can be automatically configured based on the availability of different components. This can help to better comply with the *service-level agreements* (*SLA*) established between the cloud providers and consumers. We believe that the challenge can be seen as a cooperative game between the cloud providers and consumers, that is:

1) For the service providers to comply with the service-level agreements.
2) For the consumers to receive their services with a high satisfaction rate.

As an example, we can refer to excessive spike in online shopping with "Amazon" at the end of the year. It would be helpful for both consumers and service providers if the system takes an automatic configuration strategy and relies less on busier components during certain periods. We illustrate how this can be accomplished by continuous interactions between the providers and consumers.

The good news is that, in a distributed secure system using threshold secret sharing, even if some servers do not act properly (for instance, due to an adversarial attack or delay in response time), the system can still accomplish the task if certain number of components operate appropriately. Therefore, we intend to show this cooperation can be modeled by social secret sharing. In other words, the consumers use a reputation management system to rate different components of the cloud. Subsequently, the system is reconfigured over the cloud to guarantee the service-level agreement.

Our model consists of a dealer who initiates a weighed secret sharing scheme, n cloud providers denoted by $P_1, \ldots, P_n$, and many servers interacting with the cloud providers. Let $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ and $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ be the vector of players' trust values and the vector of players' weights accordingly. The initial values in $\mathbf{r}$ are going to be zero (i.e., all service providers are treated as newcomers), whereas the initial values in $\mathbf{w}$ are chosen by the dealer based on a specific distribution. We first define the following actions where each player's action $A_i \in \{C, D, X\}$:

1) C: for *cooperative* players where P_i is available at the required time and he sends correct shares to other parties.
2) D: for *uncooperative* players where P_i

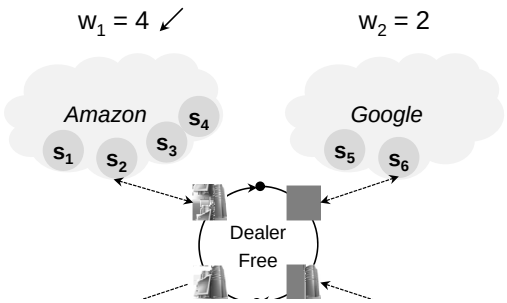

Fig. 3. Weight Adjustment

Based on the service providers' actions $A_i \in \{C, D\}$ as well as a trust function, these servers rate each component of the cloud in terms of its response time; this issue is going to be more critical in real-time systems where "response time" plays an important role. Consequently, the weight of each service provider is changed according to his new trust value. For instance, as shown in Figure 3, the weights of two components are going to be updated. To see how amplification or reduction of a trust value affects the weight of a player, see [16] for *trust-to-share ratio* computation.

In the case of corruption $A_i = X$, the corrupted providers are first rebooted. They then return to the scheme and are treated as newcomers. As we illustrated earlier, corrupted actions (e.g., sending incorrect shares) are detectable by using a verifiable secret sharing scheme.

In the final phase, the service providers jointly collaborate to reconfigure the scheme according to new weights, shown in Figure 4. They initially enroll the new shares by using an *enrollment* protocol, e.g., suppose share s_{14} is enrolled for the fourth party. Subsequently, shares are updated (except the shares that are scheduled to be disenrolled) such that they are transformed to a new secret sharing polynomial, e.g., suppose share s_4 is not updated. Hence, the first player is going to have three shares afterward.

The benefit of using threshold secret sharing in a distributed secure system is its "fault-tolerance" and "availability". For instance, if one component is compromised by an adversary or he responds with delay, other participants can carry out the intented procedure. In the next section, we provide a new trust function that better fits to our model.

IV. New Trust Function

We would like to design a new trust function with social characteristics. The function that we reviewed in Section dsec5(xce1495ence,)-95enf22(,)-417-313(s843(of)-263(95en(weper)-ce1495en9

Our modified trust function, termed *"social trust function"*, is as follows, using the previous (x) and $'(x)$ functions:

$$T_i(p) = \begin{cases} T_i(p \quad 1) + (1 \quad \frac{}{n}) \ (x) & \text{if } `_i = 1 \\ T_i(p \quad 1) \quad (\frac{}{n}) \ '(x) & \text{if } `_i = 0 \end{cases}$$

where $= \sum_{i=1}^{n} `_i$. By using the same (x) function for trust amplification and reduction in the case of cooperation and defection, the trust function can be simplified as follows:

$$T_i(p) = T_i(p \quad 1) + (`_i \quad \frac{}{n}) \ (x):$$

An example of the new social trust function is provided in Table II for further clarification. Each time the players "gain" partial of their rewards (e.g., 25%) that is proportional to the number of "non-cooperative" players. On the other hand, they "lose" partial of their trust value (e.g., 75%) that is proportional to the number of "cooperative" players.

$= \sum_{i=1}^{n} `_i$	**Cooperation**	**Defection**
n	$\mathcal{T}_i(p-1)$	no defection
$\frac{3}{4}n$	$\mathcal{T}_i(p-1) + 0{:}25 \ (x)$	$\mathcal{T}_i(p-1) - 0{:}75 \ '(x)$
$\frac{1}{2}n$	$\mathcal{T}_i(p-1) + 0{:}5 \ (x)$	$\mathcal{T}_i(p-1) - 0{:}5 \ '(x)$
$\frac{1}{4}n$	$\mathcal{T}_i(p-1) + 0{:}75 \ (x)$	$\mathcal{T}_i(p-1) - 0{:}25 \ '(x)$
0	no cooperation	$\mathcal{BT}$

$\mathcal{T} \quad \in \quad \in \ \mathcal{T} \quad \mathcal{T} \ \mathcal{T} \quad \mathcal{T} \quad \in$

[20] K. Peng, C. Boyd, E. Dawson, and K. Viswanathan, "Five sealed-bid auction models," in *the Australasian Information Security Workshop Conference, AISW'03*, vol. 21. Australian Computer Society, 2003, pp. 77–86.

VI. Appendix

A. Example of Threshold Secret Sharing

Example 3: The dealer selects secret sharing polynomial $f(x) = \mathbf{5} + 3x + 6x^2 \ 2 \ \mathbb{Z}_{13}[x]$. He then distributes the following shares among $P_1; P_2; P_3; P_4$ and leaves the scheme:

$$f(1) = 1; f(2) = 9; f(3) = 3; f(4) = 9$$

At least three players, say $P_1; P_2; P_3$, can pool their shares to recover the secret by Lagrange interpolation as follows:

$$f(0) = \frac{2}{2\ 1}\ \frac{3}{3\ 1}(1) + \frac{1}{1\ 2}\ \frac{3}{3\ 2}(9) + \frac{1}{1\ 3}\ \frac{2}{2\ 3}(3) = 21 \quad 5 \mod 13$$

B. Example of Proactive Secret Sharing

Example 4: Suppose the original secret sharing polynomial is $f(x) = \mathbf{3} + 4x + 7x^2 + 5x^3 \ 2 \ \mathbb{Z}_{13}[x]$. Players $P_1; P_2; P_3; P_4$ receive the following shares from the dealer accordingly, as shown in Figure 5:

$$f(1) = 6; f(2) = 1; f(3) = 5; f(4) = 9$$

The players securely generate $g(x) = \mathbf{0} + 4x + 2x^2 + 10x^3$ in the absence of the dealer with the following shares:

$$g(1) = 3; g(2) = 5; g(3) = 1; g(4) = 12$$

Each P_i locally adds his shares together. As a result, the new polynomial will be $\hat{f}(x) = \mathbf{3} + 8x + 9x^2 + 2x^3$ with shares:

$$\hat{f}(1) = 9; \hat{f}(2) = 6; \hat{f}(3) = 6; \hat{f}(4) = 8$$

Fig. 5. Proactive Secret Sharing