Hindawi
Security and Communication Networks
Volume 2020, Article ID 9428457, 10 pages
https://doi.org/10.1155/2020/9428457

Research Article

Broadcast Complexity and Adaptive Adversaries in Verifiable Secret Sharing

Seyed Amir Hosseini Beghaeiraveri,[1] Mohammad Izadi,[1] and Mohsen Rezvani [2]

[1]*DISYS Lab, Computer Engineering Department, Sharif University of Technology, Tehran, Iran*
[2]*Faculty of Computer Engineering, Shahrood University of Technology, Shahrood, Iran*

Correspondence should be addressed to Mohsen Rezvani; mrezvani@shahroodut.ac.ir

Received 31 July 2019; Revised 21 June 2020; Accepted 8 July 2020; Published 1 August 2020

Academic Editor: Huaizhi Li

Verifiable secret sharing (VSS) is one of the basic problems in the theory of distributed cryptography and has an important role in secure multiparty computation. In this case, it is tried to share a confidential data as secret, between multiple nodes in a distributed system, in the presence of an active adversary that can destroy some nodes, such that the secret can be reconstructed with the participation of certain size of honest nodes. A dynamic adversary can change its corrupted nodes among the protocol. So far, there is not a formal definition and there are no protocols of dynamic adversaries in VSS context. Also, another important question is, would there exist a protocol to share a secret with a static adversary with at most 1 broadcast round? In this paper, we provide a formal definition of the dynamic adversary. The simulation results prove the efficiency of the proposed protocol in terms of the runtime, the memory usage, and the number of message exchanges. We show that the change period of the dynamic adversary could not happen in less than 4 rounds in order to have a perfectly secure VSS, and then we establish a protocol to deal with this type of adversary. Also, we prove that the lower bound of broadcast complexity for the static adversary is (2,0)-broadcast rounds.

1. Introduction

In the family of distributed cryptography problems, secret sharing is a fundamental problem in which a "dealer" D in a synchronized message-passing distributed system tries to share the secret data s between a set of n players, such that every player gives his own share and, after that, every set of at least $t+1$ players could reconstruct s by combining their shares, but no set of at most t players will be able to achieve s [1].

Verifiable secret sharing (VSS) is an extended version of secret sharing, such that there is an active and external corrupter mechanism in the system [2]. Such a corrupter, which is named adversary, is capable of corrupting some of the players in an arbitrary way. In the VSS problem, even with such an adversary, reaching to secret sharing is on demand. The adversary implies an essential role in the VSS problem and also several types are defined for it. Most important deviation of adversaries is a static adversary which selects its corrupted players before the algorithm started. On the other hand, an adaptive adversary is able to adaptively corrupt players as the algorithm goes on [3].

For solving the VSS problem, the system has been considered to have two independent message-passing channels. The first way is authenticated, private channels that connect players pairwise; these channels construct a point-to-point network. In addition, there is a common broadcast channel to which all players have access. The broadcast channel allows each player to send a message to all other players consistently, even if the sender is corrupted by the adversary [4]. The broadcast channel could exist either by simulating on the point-to-point channels or by considering a "physical broadcast channel." But it should be considered that when more than a third of the players are corrupted, the broadcast channel cannot be simulated on point-to-point channels [5].

VSS protocols work in two different phases. The first phase in which the secret is shared by the dealer and each player receives his share is called *Sharing* phase. The second phase, in which players come together and try to retrieve the secret by combining their shares, is named *Reconstruction* phase. In the general view, the rounds that VSS protocols take to reach the goal in each two phases form the complexity and efficiency of the protocol.

VSS has played an important role in Data Privacy Science for the last two decades [6]. VSS is the most important subroutine of secure multiparty computation (SMC). SMC is typically accomplished by making secret shares of the inputs and manipulating the shares to compute some function in a distributed system [7]. Another application of VSS is management of financial digital signatures, which needs more than one individual's keys to sign a document. In such a case, each of the individuals must have a contribution (a share) of the master key (the secret) so they could construct the master key even in the presence of malicious efforts [1]. These types of protocols have been widely used in E-voting schemes [8].

1.1. Prior Works. Secret sharing is formally defined in [1, 9] as a two-phase algorithm (*Sharing* and *Reconstruction*). Shamir introduced an interesting method for sharing a secret s among n players with the help of polynomials and interpolation [1]. His method was not fully secured against an active adversary. In 1985, Chor et al. proposed a new method for VSS to achieve simultaneousness in asynchronous networks [2]. Their algorithm was not based on polynomials and its message and round complexity was $O(2^t)$, where t is the number of corrupted players in the system.

Next, in 1989, Rabin and Ben-Or in [7] introduced the Weak Secret Sharing (WSS) concept, which is a VSS algorithm with weaker commitment requirement. They also proposed the information checking (IC) scheme to establish a platform for information exchange between players. This scheme is based on sending information from a sender S to a recipient R via an intermediate I and is employed in many proceeding algorithms as a black box.

Genaro et al. in [4] proved the lower bounds on round complexity of VSS and WSS algorithms. They showed that every completely secure VSS algorithm with an active adversary that could be able to corrupt players needs at least three independent rounds. More importantly, there is no completely secure algorithm with a constant number of rounds to solve the VSS problem when the number of corrupted players is more than $n/3$ players. Other details of round complexities are reported in Table 1. Fitzi et al. in [10] established a tight protocol for the case where $3t < n \leq 4t$. Their protocol is completely secure and also is efficient.

Recent studies focused on reducing the number of rounds using the broadcast channel. This measurement leads to the "broadcast complexity" termination, which was introduced by Garay et al. in [11]. They considered the broadcast channel as an expensive resource and tried to reduce the number of rounds in which players have used a broadcast channel. They proposed a (2,0)-broadcast and linear protocol (2 round broadcast uses in the sharing phase and no broadcast use in the construction phase) for a system with the honest majority but with more than a third of corrupted players $(n/3) \leq t < (n/2)$. The total complexity of their algorithm was (20, 1) meaning 20 rounds in the *Sharing* phase and one round in the *Reconstruction* phase.

Table 1: Round complexities for secure VSS and WSS protocols with an active adversary.

Protocol	Threshold	Number of rounds
WSS	$n \leq 3t$	—
	$3t < n \leq 4t$	3
	$4t < n$	1
VSS	$n \leq 3t$	—
	$3t < n \leq 4t$	3
	$4t < n$, $n > 1$	2
	$4t < n$, $n = 1$	1

There are also other important algorithms with a constant broadcast complexity for $t < n/2$. Kumaresan et al. in [12] introduced a (2,2)-broadcast and (3,2)-round algorithm but their algorithm is not linear and also is in exponential time complexity. Hirt and Raykov in [13] proposed a (1,0)-broadcast protocol in which the overall number of their protocol rounds is linear in n.

Whether there is a (1,0)-broadcast and constant round protocol for the system with $t < n/2$ or not is an open problem. Also, in all of the papers presented in this section and to the best of our knowledge, the system model was based on the static adversary and there was no study on the adaptive hostile factor. In this paper, we intend to cover the gap of an adaptive adversary by providing a formal definition and a proper protocol for the adaptive adversary.

Recent efforts focused on a particular scheme of the VSS problem, which is known as the publicly verifiable secret sharing (PVSS). This form of VSS allows any third-party evaluator (outside $\mathbb{P}$) to evaluate the correctness of the node's shares. In 2014, Jhanwar et al. [14] have developed a PVSS scheme using the Paillier additively homomorphic public key encryption algorithm. The PVSS scheme has some advantages over VSS [15]. The obvious benefit of using PVSS is that anyone, even a node outside the original system, can verify the shares of each player. Also, the number of sent messages is much lower compared to traditional VSS schemes. On the other hand, PVSS does not support multishare capability, and, more importantly, it requires much more processing abilities in the players due to the use of heavy public key algorithms. In 2018, Lin et al. in [15] introduced a new publicly verifiable multisecret sharing (PVMSS) scheme, which allows multisecret sharing and needs less processor requirement.

In comparison with the abovementioned research that worked on a different form of VSS problem and considered a static adversary, the main focus of our work is on a VSS protocol for the adaptive adversaries, which needs a very low process requirement of players.

Between the newest results that are strongly focused on cloud environments, we could refer the readers to a survey

written by Attasena et al. in 2017 [16]. Deryabin et al. introduced a short sharing scheme for multicloud storage based on the Residue Number System (RNS) [17]. Also, using elliptic curves and Chinese Reminder Theorem, Sheikhi-Carjan et al. [18] proposed a new protocol that does not need a secure and private channel between the dealer and the players. In 2019, Dehkordi and Oraei [19] presented a novel verifiable multisecret sharing (VMSS) scheme that benefits from graded encoding schemes.

1.2. Our Contribution. In this paper, we delve into some open problems in the VSS problem, which are most related to the broadcast complexity and adaptive adversaries. Our motivation comes from an open question about regimes with $(n/3) \leq t < (n/2)$ in the presence of an adaptive adversary (except of the static adversary mentioned in [11] and also finding the lower bound of the broadcast complexity in the same regime with an static adversary. It is to be noted that there is no completely secure protocol for such systems, as shown in Table 1, but a negligibly small error probability is considered in such protocols [7, 20, 21].

In order to address the above-mentioned challenges, we first propose a new formal definition for an adaptive adversary based on the abilities of such an adversary. This definition helps us to demonstrate the main ability of an adaptive adversary versus the static one, which is the capability of corrupting a different set of players during the algorithm rounds. Instead of the static adversary, which selects its corrupting nodes before the algorithm starts and cannot change the corruption set, an adaptive adversary may change its set of nodes after a particular number of rounds, termed as "change-period." After arranging a proper formal definition, we prove a lower bound on the change-period parameter.

Further, given the above-mentioned formal definition, we suggest the first optimal protocol that delves with the adaptive adversary. In fact, we arrange a two-round subprotocol with one broadcast round, which could be added to any of the previous static adversary protocols. More specifically, we added this subprotocol to the Garay (2,0)-broadcast protocol that works for $(n/3) \leq t < (n/2)$ static adversary regime and turn the Garay protocol to an adaptive adversary algorithm. We validate the performance of our protocol by simulation. Our simulation results illustrate that our protocol is efficient in terms of the runtime, the memory usage, and the number of message exchanges. In fact, as the number of players increases linearly, all these three criteria grow linearly in both the original protocol for a static adversary model and our proposed protocol for an adaptive adversary.

Finally, we prove a fundamental lower bound on the number of rounds that every protocol in the presence of a static adversary needed to take to satisfy the VSS requirements. More precisely, we demonstrate that, with a static adversary, every VSS protocol must use the broadcast channel in at least two rounds.

2. Models, Tools, and Definitions

In this section, we describe all the definitions and preliminaries needed in the rest of the paper.

2.1. Network and Players. We assume a distributed message-passing synchronous system, consisting of a set of n players $\mathbb{P} = \{P_1, P_2, \ldots, P_n\}$, where all players were fully connected together via private and secure point-to-point channels. There is also a common broadcast channel (like a bus) to which all players have access. The main property of this channel is that every time a player sends a message with this bus, the message is delivered to all players forcibly and concurrently. In addition, we marked one of the players, say P_1, as the Dealer D. Players in the beginning are completely healthy and follow the protocol correctly. We named such a player an honest player.

The algorithm consists of several rounds that are a period of time through which the players send and receive a batch of data messages. In each round based on the algorithm, players send proper data to each other, using mentioned channels. When each round is finished, players perform their own computation on received data and prepare the data packets which should be sent in the next round.

For example, in an electronic and cryptographic voting system, each voter is considered as a player and each player is connected to all the others pairwise with fiber lines. In addition, we have a common microwave bus, which is used by players. For collecting votes, players send and receive authentication data in sequential rounds.

2.2. Adversary. As mentioned earlier, the properties of an adversary imply an essential role in VSS protocols. An adversary can take control of any player and turn it into a corrupted player that we call dishonest. We consider a centralized adversary $\mathcal{A}$, denoted as a *t-adversary*, meaning that the number of players which such adversary can corrupt is at most t players. The set of players which such adversary can corrupt is called adversary structure Γ.

Our common assumption of the adversary over the whole paper is that we consider that the adversary is *active* (i.e., it can force each player $P_i \in \Gamma$ to deviate from the protocol in an arbitrary way) and is *rushing* (i.e., it has access to the messages and broadcasts of each honest player in the round r before deciding on messages and broadcasts of dishonest players at the same round r).

In all sections except Section 5, we considered the adversary to be *computationally unbounded*. Also, for Sections 3–5, we assume that the adversary is *adaptive*; it means that such adversary could adaptively corrupt players as algorithm proceeds. It allows an adversary to change the number of the players under his control (but it is a *nonmobile* adversary; i.e., the number of different players of Γ is at most t). However, in Section 7 we consider a static adversary where the players of Γ are chosen before the algorithm gets started and it is not able to change corrupted players in the whole

period of the algorithm's computation. We consider the case of $(n/3) \leq t < (n/2)$. In such a regime, we are looking for statistical security rather than perfect security because perfect security is not available when $t < t/3$.

2.3. Verifiable Secret Sharing. Although there are several definitions of VSS [10, 22, 23], in all of them, protocols consist of two independent phases; we call the first phase *Sharing* phase; the dealer distributes a secret data s among all players and when the first phase is finished, the second phase starts and players come together and cooperate for retrieving the secret. Formally, we describe the above phases as the following subprotocols [10]:

(i) VSS-Share: initially, the dealer holds the secret $s \in \mathbb{A}$, where $\mathbb{A}$ is a finite set of fields with a sufficient number of elements. The dealer distributes the secret among players such that, at the final, each player P_i receives a share v_i derived from secret s.

(ii) VSS-Reconstruct: at first, each player P_i reveals some data according to the protocol about its input v_i to all other players. We named this revealed information as v_i' (clearly, a dishonest player may send some incorrect information such that $v_i' \neq v_i$). Then, based on the revealed information v_i', all players apply a reconstruction function $s = Rec(v_1, v_2, \ldots, v_n)$ to retrieve the secret s.

After the above two phases, the following three requirements of the VSS problem have to be satisfied for a correct protocol [22]:

(i) PRIVACY: if D is honest, then adversary $\mathscr{A}$ gains no information about secret s (in other words, his view is statistically independent from s) at the end of the sharing phase.

(ii) CORRECTNESS: if D is honest, then the reconstructed value, obtained at the end of the reconstructed phase, must be equal to secret s.

(iii) COMMITEMENT: with high probability, at the end of the sharing phase, there must exist a unique value $s^* \in \mathbb{A}$ based on a joint view of honest players such that, at the end of the reconstructed phase, all players output the value $s^* = Rec(v_1, v_2, \ldots, v_n)$. If D is honest, then there must exist $s^* = s$.

With the formal definition of the VSS problem, we can go through our contributions to this problem.

3. Formal Definition of Adaptive Adversary

In this section, we propose a formal definition of an adaptive adversary in VSS problems. This definition helps us to formulate abilities of an adaptive adversary in order to arrange a suitable protocol and enables us to formally prove the correctness, privacy, and commitment requirements of the VSS problem. To the best of our knowledge, there is no formal specification for the characteristic of adaptive adversaries and all we know is from properties of such adversaries in a multiparty computation problem, which is the most important application of the VSS problem [24].

We should consider that an active adversary (either static or adaptive) does not reveal his plan about corrupted players or even which player he wants to corrupt. But, to specify adaptive adversaries, we consider a set of players which an adversary can potentially corrupt in the whole period of protocol, named *adversary structure* Γ. Also, the main property of adaptive adversaries is that they can adaptively change their underlying players as protocol proceeds. We classify all of these elements in the following definition.

Definition 1 (adaptive adversary). An adaptive adversary $\mathscr{A}_P$ is represented by a Vicissitude function $\mathscr{V}$ such that

$$\mathscr{V} := \gamma \times Q \times T \longmapsto \gamma \times Q, \tag{1}$$

where γ is a forbidden set which is a subset of adversary structure Γ:

$$\gamma \subset \Gamma = \left\{B \in P \mid \mathscr{A}_p \text{ can corroupt } B\right\}. \tag{2}$$

Also, $T = m \cdot Round$ is a change period and $Q = \{q_0, q_1, \ldots, q_l\}$ is a finite set of situations in which $\mathscr{A}_p$ could change its underlying players.

In the above definition, adversary structure $\Gamma$is similar to the definition of unbounded adversary according to [3]. For example, if the system includes $P = \{P_1, P_2, \ldots, P_{20}\}$, the following set could represent an instance of adversary structure:

$$\Gamma = \{\{P_4, P_5, P_{10}\}, \{P_1, P_3, P_4\}, \{P_{15}, P_{20}, P_{14}, P_8\}, \{P_7, P_{19}\}\}. \tag{3}$$

$\mathscr{A}_p$ could choose each of sets from Γ in the beginning and continues while changing its situation by choosing another set adaptively. Here, the change period concept takes the role. In fact, the change period is the number of rounds that $\mathscr{A}_p$ should wait for changing the set of players under his control. The minimum value of the change period is one round as no transformation can essentially happen in less than one round in a synchronous system. Also, the change period always is a multiple of one round and guarantees the mobility of an adaptive adversary.

In Definition 1, we also have a set of situations where $\mathscr{A}_p$ could change its corresponding players. When this set of situations is finite, the Vicissitude function could be presented as a finite state machine. In the above example, the final representation of our adversary could be as follows:

$$\mathscr{V} = \begin{cases} q_0\colon \left(\{P_{15}, P_{20}, P_{14}, P_8\},\ 2 \cdot Rounds\right) \longmapsto q_2, \\ q_1\colon \left(\{P_1, P_3, P_4\},\ 1 \cdot Rounds\right) \longmapsto q_2, \\ q_2\colon \left(\{P_4, P_5, P_{10}\},\ 3 \cdot Rounds\right) \longmapsto q_1. \end{cases} \tag{4}$$

The adversary structures could have different properties, but, in the following, we define the $t - bounded$ condition used in the rest of the paper:

Definition 2 ($t - bounded$ condition). An adversary structure Γ satisfies the $t - bounded$ condition if none of its subsets have more than t members.

$$\forall B \in \Gamma: |B| \leq t. \quad (5)$$

This circumstance brings the static t-adversary into mind. In fact, if we take a snapshot from an adaptive adversary, due to the protocol, the t-bounded adaptive adversary is similar to a static t-adversary.

In this section, we defined a formal definition of an adaptive adversary. Defining in this way empowers us to suggest a protocol with respect to the adversary behavior which informal definition is not able to.

4. Lower Bound of Change Period

It is significant to know how much an adaptive adversary can make problems in VSS protocols. In better words, an adaptive adversary who is too fast in changing is theoretically irreparable and it is needed to find an acceptable changing rate for challenging such an adversary. This section is about a lower bound on change period that, with a number lower than such period, there is no perfectly secure protocol to solve a VSS problem.

Theorem 1 (change period's lower bound). *There is no perfectly secure VSS protocol in the presence of a rushing, computationally unbounded, and adaptive t-bounded adversary with a change period of less than four rounds.*

Proof. Consider that there is an Algorithm 1 that can solve the VSS problem with perfect security with three-round adaptive adversary (consumption) in a system with honest majority. Because the change period of the adversary is three rounds, it could change its players after round three. Assume that, at first, B_1 is the forbidden set and, after three rounds, the adversary changes its forbidden set to B_2. Without loss of generality, it could be happening that $|B_1 \cup B_2| \geq t + 1$ (e.g., B_1 has exactly t items and B_2 has at least one distinctive item from B_1). Since the adversary is rushing, it can access the messages of all corrupted players in B_1 at the beginning of the third round and it could save all of these messages (because the adversary is computationally unbounded). After change in the forbidden set, the adversary has also access to messages of players in B_1. Totally, the adversary owns at least $t + 1$ players and has access to their information. Clearly, we can replace this adaptive adversary with a static $(t + 1)$-adversary in a (3,0)-round protocol. According to Table 1, solving the VSS problem with honest majority needs at least three rounds with a t-adversary. But we have a $(t + 1)$-adversary, which is able to solve the VSS problem. If we take $t = n/3$, this is a contradiction and so there is not such an algorithm. □

5. Subprotocol for Computationally Bounded Adaptive Adversary

In Theorem 1, we suppose that the adversary is computationally unbounded. On the other hand, if the adversary has an upper bound on its computational power, for example, on its internal memory, it cannot access the messages of all players in the forbidden set so an alternative protocol can fight with this adversary properly.

For example, consider that our adaptive adversary is able to keep the only $t < n$ different data packet in each round. Such a condition would be approximately weak (but the adversary still uses an unbounded arithmetic, logic, and mathematic computational power). Now when the forbidden set gets changed by the adversary, he can hold at most t different shares of players. However, using a proper protocol like [11], the privacy of the algorithm is satisfied because the adversary has access to at most t different shares.

Unlike the privacy requirements, since the corrupted players may send incorrect data packets to other players, the correctness of the algorithm is still in danger. Therefore, we suggest a very efficient subprotocol to simulate ordinary message sending over private channels. This subprotocol can be used in other static adversary protocols and turn them into a suitable protocol for memory-bounded adaptive adversaries. This subprotocol should recall inside of static adversary protocols everywhere players want to send data to each other over private channels. We named this subprotocol *SendWithCheck()*. Our subprotocol is based on a very simple and efficient (in time and message consumptions) integrity checking routine which uses random numbers.

> Model: we consider a system with an adaptive rushing but computationally bounded (only in memory) adversary that can corrupt $(n/3) \leq t < (n/2)$ players. In [11], a (2,0)-broadcast (20,1)-round protocol was suggested for this regime albeit with a static adversary so it is a good example to enhance. We mentioned earlier that the sharing phase of [11], which is named $VSS - Share_{2bc}$, just like any other protocol, consists of several message exchanges through private channels. Our subprotocol is called against ordinary message sending any time a player decides to send a message to another player. We prove that if we use the *SendWithCheck()* subprotocol (shown in Algorithm 1) as a message transition protocol, the adversary is not able to threat the correctness of the $VSS - Share_{2bc}$ protocol.

The *SendWithCheck()* routine is a simple information checking protocol based on the integrity measurement of the sender and examination with putting the dealer as an intermediary. In other words, for each message transmission, dealer generates two independent random numbers and sends one of them to the sender and gives information about both of them to the receiver. The sender P_j has to send the only correct information to a receiver in order for his message to get matched with the receiver information; otherwise, the receiver detects the incorrect message. In following paragraphs, with accepting that the main $VSS - Share_{2bc}$ satisfies the VSS requirements, we prove that the *SendWithCheck()* subprotocol satisfies the VSS requirements, most importantly the correctness, in the presence of an adaptive adversary.

Theorem 2 (perfect secure VSS protocol for memory-bounded adaptive adversary). *Using SendWithCheck()*

Inputs. Player P_i, Player P_j, Dealer D, Message m
Goal. To deliver message m from player P_j to player P_i with the presence of an adaptive, rushing but computationally bounded (only in memory) adversary.

(1) Private channel
  (a) P_j sends message m to D.
(2) Broadcast
  (a) D chooses two random pad r_i, r_j.
  (b) D sends r_j to P_j.
  (c) D sends r_i and $\beta_{ji} = r_i + r_j + m$ to P_i.
  (d) P_j sends m and $\alpha_{ji} = r_j + m$ to P_i.
  (e) [Internal computations:] P_i computes the value $\beta_{ji} - \alpha_{ji}$. If the value is not equal to r_i, P_i rejects the message m, otherwise, P_i accepts the message m.
  (f) P_i broadcasts rejecting or accepting message m plus the sender P_j. Each player holds the set $REJECT$ which initially is empty and keeps every P_j which its message rejected. If $|REJECT| \geq t + 1$ then D disqualified.
(3) In $VSS - Share_{2bc}$ protocol
  (a) Each player holds the set $UNHAPPY$ which initially is empty. After distributing the shares (first round of $VSS - Share_{2bc}$) each player assigns $UNHAPPY := UNHAPPY \cup REJECT$. If $|UNHAPPY| \geq t + 1$ then the secret s could not be reconstructed and players have to commit on alternative value $s^* \in \mathbb{A}$.

Algorithm 1: SendWithCheck.

subroutine as player's transmission protocol in $VSS - Share_{2bc}$ generates a perfectly secure VSS protocol with a rushing, memory-bounded, adaptive adversary.

Proof

(Privacy) Because adversary can hold at most t player's shares, it does not have enough information to retrieve secret s. In other words, this case is similar to a static t-adversary and $VSS - Share_{2bc}$ guaranties the privacy in such a situation.

(Correctness) First, consider that D is honest in the first round of $VSS - Share_{2bc}$. During the distribution of shares, private channels are used in $VSS - Share_{2bc}$ and we know at most t players are corrupted and at most t rejection will accrue. Then with $n - t$ correct shares, players are able to retrieve the secret. If D breaks in another round, it can be detected by the *SendWithCheck()* subprotocol and correctness is not a problem.

(Commitment) First, we assume that D is honest. Honesty of D means that, due to *SendWithCheck()* running, if a dishonest sender sends an incorrect message, the receiver and all other players could detect the error and put the dishonest sender in their *REJECT* list. Since adversary is t-bounded, in each snapshot system cannot have more than t errors in the sent information. Also, if receivers are dishonest and deviate from the protocol, at most t players disqualify the dealer and D is recognized as an honest player. If D remains honest in the whole period of the algorithm, all players have $n - t$ correct shares and, with the acceptance of the commitment of $VSS - Share_{2bc}$, all players commit on the secret s.

On the other hand, if D was dishonest initially or breaks due to the protocol, there is a possibility that it makes more than t errors with generating bad random numbers or incorrect messages, while in reality at most t errors can happen. Since players check this situation in their internal computations and also hold dishonest player's ID in their *REJECT* list, if D breaks in even one round, players detect it and try to retrieve an alternative s^* which is possible in $VSS - Share_{2bc}$ protocol.

As we can see, this subprotocol could detect all errors in the dealer or other players and even with an adaptive adversary. In the worst case, the change period is one round and changing in the forbidden set happens between the first and second rounds of *SendWithCheck()*. In this case, in the first round, just the sender sends m to the dealer:

(i) If sender is corrupted after the second round, the receiver rejects the sender if the sender information does not send correct messages in the second round.

(ii) If dealer breaks between two rounds and if it sends incorrect information more than t times, the receiver could detect the dishonest dealer and report that (if dealer breaks and also sends correct information, it helped to the correct running of protocol and there will be no damage). □

6. Performance Analysis

In this section, we present a performance analysis of the *SendWithCheck()* protocol. As mentioned in Section 5, *SendWithCheck()* is a subprotocol that replaces normal message sending over private channels in order to deal with a memory-bounded adaptive adversary. The purpose of this section is to examine the behavior of a real system in the two models of using the *SendWithCheck()* protocol and without using it. By comparing these two models, we can evaluate the amount of overhead created to deal with an adaptive adversary.

6.1. Scenario. A number of players in a secret sharing system plan to share a secret in the model described in Section 3 by dealer. Since the *SendWithCheck()* protocol can be used in the sharing phase of every secret sharing protocol with a

static adversary, we consider $WSS - Share_{2bc}$ secret sharing protocol introduced in [11] for implementation. At each stage of the experiment, players perform the $WSS - Share_{2bc}$ protocol once without the *SendWithCheck()* and again with this subprotocol. This experiment is conducted to measure the number of messages exchanged, the amount of time it takes to complete $WSS - Share_{2bc}$, and finally the total memory consumed by the players. All other conditions of the protocol, such as the adversary factor and internal calculations, are assumed to be constant throughout all of the experiments. In the case of the adversary, it does not actively intervene during the execution of the protocol, and the protocol is assumed to be fixed feedback from the adversary. Also, all internal calculations are considered the same for all players in all situations.

6.2. Simulation Environment. The player network simulation was performed using the Swift 5.1 programming language. The program source code is accessible at https://github.com/seyed1411/WSS_SendWithCheck.git. In this code, two types of players are defined: Participant and ParticipantSWC. The former player type is defined without the *SendWithCheck()* subprotocol, while the latter type is using this subprotocol; each runs the WSS sharing phase protocol of [11]. Since our subprotocol is only used in the sharing phase, there is no need to simulate the reconstruction phase for this comparison. The code is executed to get the number of messages exchanged, the execution time, and finally the memory consumption for growing the number of players in order to observe the scalability of our subprotocol. All experiments are performed on a PC machine with a quad-core 2.5 GHz processor and 6 GB of RAM.

6.3. Message Exchanging. As mentioned earlier, the *SendWithCheck()* protocol, using the dealer as a middle node and a number of additional message transmissions, replaces the use of the normal message exchanging from player P_j to player P_i. So, it will be interesting to see what effect this replacement has on the overall message exchanged between the players. In this experiment, taking all other fixed conditions, the players perform the $WSS - Share_{2bc}$ Sharing protocol once by running *SendWithCheck()* for each single message transfer and once by transferring the normal messages. Experiments have been repeated on a variant number of players from 5 to 500 with an increase of 50 each step to see the effect of the linear increase in the number of players on the number of exchanged messages. Figure 1 shows the results of this experiment. As we can see, with the linear increase in the number of players, both models provide a similar complexity in terms of the number of messages. In fact, as the number of players increases linearly, the number of messages on both protocols increases in a linear form.

6.4. Runtime. Replacing simple message exchange with a subprotocol like *SendWithCheck()*, which has some additional steps, can cause a significant increase in the running time of the main protocol. In this experiment, we evaluate the effect of adding *SendWithCheck()* protocol on the time required to complete a round of $WSS - Share_{2bc}$ protocol. In this experiment, players once perform the code of the protocol in the presence of *SendWithCheck()* and once again transmitted normal messages for 5 to 500 players increase by 50 in each step. Figure 2 shows the results of this experiment. As we can see, similar to the previous experiment, with the linear increase of the number of players, both models have the same growth in the execution time to complete the sharing phase. Therefore, in this experiment, as the number of players increases linearly, the runtime of both models grows nonexponentially.

6.5. Memory Consumption. In the last experiment, we investigate the effect of adding our subprotocol on the amount of memory required to run the original sharing phase. The ability to run the *SendWithCheck()* subprotocol requires the addition of some extra fields to hold subprotocol messages, as well as a number of internal variables for player's internal computations. In order to see the growth of memory consumption as the number of players increases, we measure the occupied memory of the protocol by increasing the number of players as before. Figure 3 shows the results of this experiment. As one can see in this figure, similar to the previous experiments, with the linear increase of the players, both models have the same growth in consuming system memory to complete their sharing phase. Therefore, in this experiment, as the number of players increases linearly, the memory usage of both modes increases nonexponentially.

7. Lower Bound on Broadcast Complexity with Static Adversary

As we mentioned in Section 1, the best algorithm for static adversaries in a system with $(n/3) \leq t < (n/2)$ is the (2,0)-broadcast protocol proposed by Garay et al. [11]. In this section, we prove that two broadcasts are the minimum amount of complexity we can reach in the presence of a static adversary. The basement of this proof is on the contradiction of existing (1,0)-broadcast algorithms.

The proof is based on three lemmas. First of all, we prove these lemmas that are used in the proof of broadcast complexity lower bound. Assume that there is Algorithm 1 that can achieve the VSS requirements in the presence of static adversary in one round broadcast usage.

Lemma 1. *For achieving the commitment requirement, Algorithm 1 has to distribute two nonmerging sets of information for the secret retrieving and dealer examination.*

Proof. We call the first information set "Secret Retrieving" and the second information set "Dealer Examination." Clearly, the Secret Retrieving set is shares of each player distributed by the dealer and should guarantee the existence of a fixed value for a player commitment. According to the VSS formal definition, sharing of the secret is performed by the dealer D. However, the Dealer Examination set must be

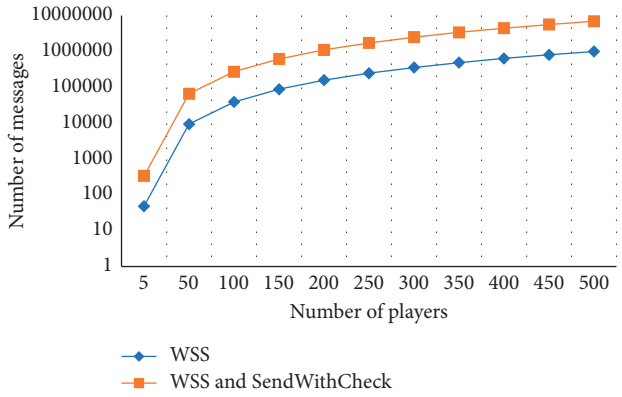

Figure 1: Growth of messages transmission with and without SendWithCheck() subprotocol.

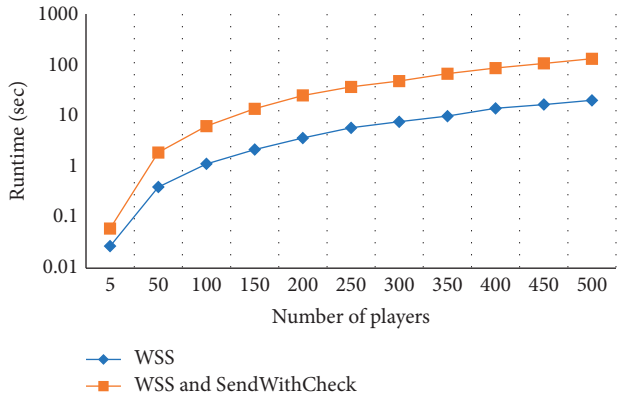

Figure 2: Growth of runtime with and without SendWithCheck() subprotocol.

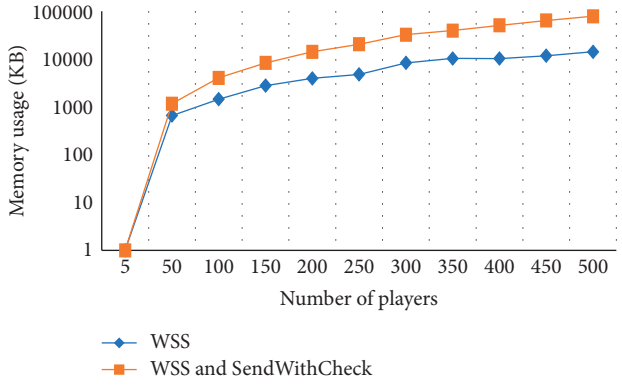

Figure 3: Growth of memory consumption with and without SendWithCheck() subprotocol.

distributed to players $\mathbb{P} \setminus D$; otherwise a corrupted D is able to distribute fake information to pretend that it is an honest dealer.

In order to achieve the commitment requirement, the dealer must send the Secret Retrieving information to players. Suppose that we can merge the Dealer Examination set into the Secret Retrieving information set. If in this case the dealer was corrupted by an adversary, Secret Retrieving information could be manipulated by the adversary such that in reconstruction phase players get apparently correct information and commit on a value $s^* \notin \mathbb{A}$ which breaks the commitment condition. Thus, it is necessary that these two information sets be distinct. □

Lemma 2. *To achieve the commitment requirement with the agreement, Algorithm 1 must distribute the Secret Retrieving information by broadcasting.*

Proof. In Lemma 1, we showed that the Secret Retrieving set is needed to be delivered to all players. Suppose that Secret Retrieving set is delivered to players with private secure point-to-point channels except broadcasting. If D is corrupted by an adversary, it might happen that D partitions the player set $\mathbb{P}$ into two player sets $\mathbb{P}_1$ and $\mathbb{P}_2$ and sends the correct Secret Retrieving information to $\mathbb{P}_1$ and incorrect Secret Retrieving information to $\mathbb{P}_2$. If there was no broadcast of the Secret Retrieving information, players of two partitions cannot be aware about differences in Secret Retrieving information. In other words, partition $\mathbb{P}_1$ can retrieve correct secret where the partition $\mathbb{P}_2$ cannot retrieve the secret s and computes $s^* \notin \mathbb{A}$. Therefore, agreement situation is not taken between two partitions of players. If we want to find disagreement and solve the difference with a broadcast message, then we take one broadcast and the lemma is proved. □

Lemma 3. *To achieve the correctness, Algorithm 1 must distribute the dealer examination information by broadcasting.*

Proof. Suppose that the Dealer Examination set is delivered to players with private secure point-to-point channels except broadcasting. If D is honest, there are t players under control of a static t-adversary that can deviate from protocol. More precisely, these corrupted players can generate some fake dealer examination information such that the honest dealer seems to be dishonest in the view of some honest players. Since this fake Dealer Examination information is sent by point-to-point channels, there is no way for honest players to check fakeness. Thus, the players disqualify the dealer in the event that the dealer is honest and players should output the secret s. Therefore, the correctness of the VSS problem is not satisfied and the dealer examination must also be distributed by broadcasting. □

Theorem 3 (lower bound of broadcast complexity). *The lower bound of broadcast complexity in the VSS problem with rushing, computationally unbounded, and static t-adversary is at least 2.*

The first principle is that every VSS protocol needs at least one use of the broadcast channel. We show that this single round is insufficient to satisfy the requirements of the VSS problem. It is obvious that the distribution of the Dealer Examination information occurs after distributing some Secret Retrieving information (because the dealer still distributes no Secret Retrieving information to players and before this examination makes no sense). Also, according to

Lemma 1, there are two independent, distinct, and nonmerging information sets that must be broadcasted according to Lemmas 2 and 3. Therefore, every VSS protocol with static t-adversary needs at least two broadcast channel usages to achieve requirements. Thus, the theorem is proved.

8. Conclusion

In this paper, we delved into some open questions about broadcast complexity and adaptive adversaries in the VSS problem. We proposed a formal definition for adaptive adversaries, helping us to characterize the circumstance of such adversary. In addition, we proved a lower bound on one of the parameters of this definition. Also, we proposed an efficient two-round (one broadcast) subprotocol which was added to protocols that are suitable for static adversaries and turn them into perfect secure VSS protocols for adaptive memory-bounded adversaries and also provide a performance analysis of this protocol. Finally, we proved an important lower bound on the broadcast complexity of VSS in the presence of a static adversary.

There are still works to do in the field of VSS problems. The most important opened window is the concept of adaptive adversaries, which needs to design more optimized protocols for this type of adversary. Indeed, the finite state form of the adaptive adversary definition provides a good potential to use different concepts of Automata theory in adaptive adversaries' protocols. It would be a good practice to decrease the number of rounds in the reconstruction phase of Garay (2,0)-broadcast protocol to a number smaller than 20.

Data Availability

The nature of the data is the Swift 5.1 language source code written for the simulation of the proposed approach. The data used to support the findings of this study are included within the article (the environment configuration of the simulation). The program source code is accessible at https://github.com/seyed1411/WSS_SendWithCheck.git.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this article.

References

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.

[2] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, New York, NY, USA, 1985.

[3] M. Hirt and U. Maurer, "Complete characterization of adversaries tolerable in secure multi-party computation," in *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, vol. 25–34, New York, NY, USA, 1997.

[4] R. Gennaro, Yuval Ishai, E. Kushilevitz, and T. Rabin, "The round complexity of verifiable secret sharing and secure multicast," in *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, 2001.

[5] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[6] M. Backes, A. Kate, and A. Patra, "Computational verifiable secret sharing revisited," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 590–609, New York, NY, USA, 2011.

[7] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, vol. 73–85, ACM, New York, NY, USA, 1989.

[8] V. P. Binu, D. G. Nair, and A. Sreekumar, "Secret sharing homomorphism and secure E-voting," 2016.

[9] G. R. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the International Workshop on Managing Requirements Knowledge*, vol. 313, IEEE Computer Society, Los Alamitos, CA, USA, 1979.

[10] M. Fitzi, J. Garay, S. Gollakota, C. Pandu Rangan, and S. Kannan, "Round-optimal and efficient verifiable secret sharing," in *Theory of Cryptography*, pp. 329–342, Springer, Berlin, Germany, 2006.

[11] J. Garay, C. Givens, R. Ostrovsky, and P. Raykov, "Broadcast (and round) efficient verifiable secret sharing," in *Information Theoretic Security, 200–219*, Springer, Berlin, Germany, 2014.

[12] R. Kumaresan, A. Patra, and C. Pandu Rangan, "The round complexity of verifiable secret sharing: the statistical case," in *Advances in Cryptology-ASIACRYPT 2010*, pp. 431–447, Springer, Berlin, Germany, 2010.

[13] M. Hirt and P. Raykov, "On the complexity of broadcast setup," in *Automata, Languages, and Programming*, pp. 552–563, Springer, Berlin, Germany, 2013.

[14] M. P. Jhanwar, A. Venkateswarlu, and R. Safavi-Naini, "Paillier-based publicly verifiable (non-interactive) secret sharing," *Designs, Codes and Cryptography*, vol. 73, pp. 529–546, Springer, Berlin, Germany, 2014.

[15] C. Lin, H. Hu, C.-C. Chang, and S. Tang, "A publicly verifiable multi-secret sharing scheme with outsourcing secret reconstruction," *IEEE Access*, vol. 6, pp. 70666–70673, 2018.

[16] V. Attasena, J. Darmont, and N. Harbi, "Secret sharing for cloud data security: a survey," *The VLDB Journal*, vol. 26, pp. 657–681, 2017.

[17] M. Deryabin, N. Chervyakov, A. Tchernykh et al., "Secure verifiable secret short sharing scheme for multi-cloud storage," in *Proceedings of the 2018 International Conference on High Performance Computing Simulation (HPCS)*, pp. 700–706, New York, NY, USA, 2018.

[18] M. Sheikhi-Garjan, M. Bahramian, and C. Doche, "Threshold verifiable multi-secret sharing based on elliptic curves and Chinese remainder theorem," *IET Information Security*, vol. 13, no. 3, pp. 278–284, 2019.

[19] M. H. Dehkordi and H. Oraei, "How to construct a verifiable multi-secret sharing scheme based on graded encoding schemes," *IET Information Security*, Institution of Engineering and Technology, New York, NY, USA, 2019.

[20] D. Chaum, C. Claude, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, 1988.

[21] D. Dolev, C. Dwork, O. Waarts, and M. Yung, "Perfectly secure message transmission," *Journal of the ACM (JACM)*, vol. 40, no. 1, pp. 17–47, 1993.

[22] A. Choudhury, K. Kurosawa, and A. Patra, "The round complexity of perfectly secure general VSS," in *Information Theoretic Security*, vol. 143–162, Springer, Berlin, Germany, 2011.

[23] J. Katz and C.-Y. Koo, "Round-efficient secure computation in point-to-point networks," in *Advances in Cryptology-EUROCRYPT 2007*, Springer, Berlin, Germany, 2007.

[24] E. Ben-Sasson, S. Fehr, and R. Ostrovsky, "Near-linear unconditionally-secure multiparty computation with a dishonest minority," in *Advances in Cryptology–CRYPTO 2012*, pp. 663–680, Springer, Berlin, Germany, 2012.