Verifiable Secret Sharing as Secure Computation

Rosario Gennaro* and Silvio Micali

Laboratory for Computer Science
Massachusetts Institute of Technology

Abstract. Verifiable Secret Sharing is a fundamental primitive for secure cryptographic design. We present a stronger notion of verifiable secret sharing and exhibit a protocol implementing it. We show that our new notion is preferable to the old ones whenever verifiable secret sharing is used as a tool within larger protocols, rather than being a goal in itself. Indeed our definition, and so our protocol satisfying it, provably guarantees reducibilty. Applications of this new notion in the field of secure multiparty computation are also provided.

1 Introduction

Secret Sharing and Verifiable Secret Sharing (VSS for short) are fundamental notions and tools for secure cryptographic design. Despite the centrality and the maturity of this concept (almost 10 years passed from its original introduction), we shall advocate that a stronger and better definition of a VSS is needed in order to achieve the very desirable property of reducibility for secure protocols. We shall then provide the first provably correct implementation of this stronger definition.

REDUCIBILITY is an essential tool for secure protocols design because it allows such protocols to be built in a *modular* way. In our case this means that one can first design a protocol P for a given task assuming that an "abstract" and "perfect" VSS protocol exists. Then one designs a correct (according to a given definition!) VSS protocol Q. Finally one substitutes Q in place of the abstract sub-protocol in P. However this way of proceeding yields a secure protocol P if and only if the definition of VSS satisfied by Q enjoys the reducibility property. Unfortunately no prior notion of a VSS provably guarantees reducibility. Thus it is a goal of this paper to provide such a definition and a VSS protocol that satisfies it.

THE INTUITIVE NOTION OF A VSS. As first introduced by Chor, Goldwasser, Micali and Awerbuch in [4], a VSS protocol consists of a two-stage protocol. Informally, there are n players, t of which may be *bad* and deviate from their prescribed instructions. One of the players, the *dealer*, possesses a value s as a secret input. In the first stage, the dealer commits to a unique value v (no matter what the bad players may do); moreover, $v = s$ whenever the dealer is honest. In the second stage, the already committed value v will be recovered by all good players (no matter what the bad players might do).

* Contact author. Email address: `rosario@theory.lcs.mit.edu`. Research supported by NSF grant no.9121466-CCR

Prior work. Several definitions and protocols for VSS have been proposed in the past ten years (E.g., [4, 1, 3, 7, 10].) We contend, however, that these notions and these protocols are of *very limited use.* In fact, their security concerns "begin when the dealer's secret is committed, and end when it is recovered." That is these notions do not concern themselves with what happens when VSS is used as a subprotocol. Because in many applications running a *single* VSS protocol is exactly what is wanted, these prior definitions and protocols are totally adequate in those scenarios. However, they are not adequate in more general scenarios since it is by now a well-known phenomenon that protocols that are secure by themselves, cease to be secure when used as a sub-protocols. In these cases the security of the entire protocol must be proven "from scratch" (for instance, this is the case in [7] where they use VSS as a tool to reach Byzantine agreement) rather than in a more natural and elegant "modular way." A notion of security that guarantees reducibility has been presented by Micali and Rogaway [9] for the problem of *function evaluation*, but not for general multiparty protocols.

Our work. We extend reducibility-guaranteeing notions of security to verifiable secret sharing protocols and concretely exhibit VSS protocols that provably satisfy these notions. More precisely, in this paper we achieve the following goals:

1. We propose a new definition of VSS casted as a special istance of secure function evaluation.
2. We compare our new notion with the previously proposed ones, and show that it is *strictly and inherently stronger.*
3. We modify earlier VSS protocols of [1, 10] and show that our new protocol is secure according to our notion.
4. Finally we present some applications that use VSS as a subroutine and need our stronger notion of VSS to be secure. These include shared authentications and signatures and a very elegant proof of the security of the completeness theorem on multiparty secure computation by Ben-Or, Goldwasser and Wigderson. [1]

2 Prior work

In order to focus on the difficulties that are proper of VSS, we shall deal with a simple computational model, both when reviewing prior work and when presenting our new one.

Computational Model. We consider n players communicating via a very convenient synchronous network. Namely, to avoid the use of Byzantine Agreement protocols we allow players to *broadcast* messages, and, in order to avoid the use of cryptography, we assume that each pair of players is connected by a *private* communication channel (i.e., no adversary can interfere with or have any access to messages between good players).

We model the corrupted processors as being coordinated by an *adversary* $\mathcal{A}$. This adversary will be *dynamic* (i.e., decides during the execution of the protocol which processors corrupt); *all-powerful:* (i.e., can perform arbitrarily long computations); and *completely-informed* (i.e., when corrupting a player she finds out all his computational history: private input, previous messages sent and received, coin tosses, etc.). Further, the adversary is also allowed *rushing* (i.e., in a given round of communication, bad players receive messages before the good ones and, based on those messages, the adversary can decide whom to corrupt next). We say that such an adversary is a t-adversary ($0 \leq t \leq n$) if t is an upper bound on the number of processors she can corrupt (t is also referred to as the *fault-tolerance* of the protocol.) This computational model is precisely discussed in [7] and [9].

PRIOR DEFINITIONS OF VSS. To exactly capture the informal idea of a VSS, has proven to be an hard task in itself. The definition reviewed below is that of [7], which relies on the notion of a *fixed event*:

Definition: We say that an event $\mathcal{X}$ is *fixed* at a given round in an execution E of a protocol, if $\mathcal{X}$ occurs in any execution E' of the protocol coinciding with E up to the given round.

Definition 1. Let P be a pair of protocols where the second is always executed after the first one, P =(`Share-Verify, Recover`). In protocol `Share-Verify`, the identity of the *dealer* is a common input to all players, and the secret is a private input to the dealer; the output of player P_i is a value $verification_i \in \{yes, no\}$. In protocol `Recover`, the input of each player P_i is his computational history at the end of the previous execution of `Share-Verify`; the output of each P_i is a string σ_i. We say P is a VSS protocol with fault-tolerance t if the following 3 properties are satisfied:

1. *Acceptance of good secrets:* In all executions of `Share-Verify` with a t-adversary $\mathcal{A}$ in which the dealer is good, $verification_i = yes$ for all good players P_i.
2. *Verifiability:* If less than t players output $verification = no$ at the end of `Share-Verify` then at this time a value σ has been fixed and at the end of `Recover` all good players will output the same value σ and moreover if the dealer is good $\sigma =$ the secret. [2]
3. *Unpredictability* In a random execution of `Share-Verify` with a good dealer and the secret chosen randomly in a set of cardinality m any t-adversary $\mathcal{A}$ won't be able to predict the secret better than at random i.e. if $\mathcal{A}$ outputs a number a at the end of `Share-Verify` then $Prob[a = s] = \frac{1}{m}$

SECURE COMPUTATION. Let us summarize the definition of secure function evaluation of [9]. Informally the problem is the following: n players $P_1, \ldots, P_n$, holding, respectively, private inputs $x_1, \ldots, x_n$, want to evaluate a vector-valued function f on their individual secret inputs without revealing them (more than already implied by f's output). That is, they want to compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ such that each player P_i will learn exactly y_i.

This goal is easily achievable if there is an external and trusted party, who privately receives all individual inputs and then computes and privately hands out all individual outputs. Of course, even in this *ideal* scenario, the adversary can create some problems. She can corrupt a player P_i before he gives his input x_i to the external party and change it with some other number $\hat{x_i}$. And she can still corrupt players after the function has been evaluated and learn their outputs. These problems should, however, be regarded as *inevitable*. Indeed, following [8], [9] call a protocol for evaluating f secure if it

[2] Notice that if we simply ask in the Verifiability condition that "all the good players output the same number σ at the end of the `Recover` phase" it would not be sufficient for our purposes. In fact, we would still allow the adversary to decide during `Recover` what value σ the good players will output. Thus `Share-Verify` would not model a secret commitment as required.

approximates the above ideal scenario "as closely as possible." The nature of this approximation is informally summarized below.

Definition *(Initial configuration, traffic, input and output):* Let us define the following quantities within the context of a protocol P.

The *initial configuration* for P is a vector $\mathbf{ic}$, whose ith component, $ic_i = (x_i, r_i)$ consists of the private input and the random tape of player P_i.

The *traffic* of player P_i in protocol P at round q, t_i^q, is the set of messages sent and received by P_i up to that round.

A *local input function* $\mathbf{I} = (I_1, \ldots, I_n)$ for P is an n-tuple of functions such that there exists a specific round r such that, by applying I_i to the traffic t_i^r, we get the input player P_i is "contributing to the computation." $\mathbf{I}(\mathbf{ic})$ will denote the vector of those values when P is run on initial configuration $\mathbf{ic}$.

A *local output function* $\mathbf{O} = (O_1, \ldots, O_n)$ for P is an n-tuple of functions such that by applying O_i to the final traffic t_i^{final} of player P_i we get his output.

Definition (*Adversary view*): The *adversary view*, $VIEW_{Network}^{\mathcal{A}}$, during P is the probability distribution over the set of computational histories (traffic and coin tosses) of the bad players.

Definition *(Simulator and ideal evaluation oracle):* A simulator Sim is an algorithm that "plays the role of the good players". The adversary interacts with the simulator as if she was interacting with the network. The simulator tries to create a view for the adversary that is indistinguishable from the real one. He does this without knowing the input of the players, but it is given access to a special oracle called the *ideal evaluation oracle*. For a protocol P with local input function $\mathbf{I}$ evaluatable at round r, the rules of the interaction between Sim and the oracle are the following:

- if $\mathcal{A}$ corrupts player P_i before round r Sim gets from the oracle the input x_i and gives it to $\mathcal{A}$.
- at round r Sim gets the output y_i' for all the players corrupted so far, where $(y_1', \ldots, y_n') = f(x_1', \ldots, x_n')$ where $x_i' = x_i$ if P_i is still good, otherwise $x_i' = I_i(t_i^r)$
- if $\mathcal{A}$ corrupts a player P_i after round r, Sim gets from the oracle the pair (x_i, y_i') and gives it to $\mathcal{A}$.

Definition 2. *Secure Function Evaluation:* Let f be a vector-valued function, P a protocol, Sim a simulator, and $\mathbf{I}$ and $\mathbf{O}$ local input and output functions. We say that P securely evaluates the function f if

- *Correctness:* If $\mathbf{ic}$ is the initial configuration of the network, then
  1. $x_i = I_i(\mathbf{ic})$ for all good players P_i
  2. with high probability, $\mathbf{O}(\mathbf{t}^{final}) = f(\mathbf{I}(\mathbf{ic}))$

  (I.e. no matter what the adversary does, the function is evaluated during the protocol on some definite inputs defined by the local input functions over the traffic of the players. These inputs coincide with the original inputs for the good players)
- *Privacy:* For all initial configurations $\mathbf{ic}$, if $VIEW_{Sim}^{\mathcal{A}}$ is the adversary view of the simulated execution of the protocol, we have that

  $$VIEW_{Network}^{\mathcal{A}} \approx VIEW_{Sim}^{\mathcal{A}}$$

  (I.e., the two views are statistically indistinguishable.)

There are many reasons for which this definition captures correctly the notion of a secure computation. In particular, the following one: the definition in [9] allows one to prove formally many desirable properties of secure protocols, the most interesting for us being *reducibility*:

Theorem 3 [9]. *Let f and g be two functions. Suppose there is a protocol P that securely evaluates f in the model of computation in which it can perform ideal evaluations of g. Suppose also that there is a protocol Q that securely computes g. Denote with P^Q the protocol in which the code for Q is substituted in P in the places where P ideally computes g. Then P^Q is secure.*

Interested readers are referred to the original paper [9] for a proof of this statement and a complete and a formal description of their definition.

3 Our definition of VSS

In this section we provide a new definition of VSS that guarantees reducibility. The key idea for achieving this property is to cast VSS in terms of secure function evaluation. Accordingly, we shall define two special functions SHAR and REC, and demand that both of them be securely evaluated in the sense of [9].

We assume a network of n players $P_1, \ldots, P_{n-1}$ and P_n, where $P_n = D$ the dealer. Let Σ be a set. Consider the vector space Σ^n and the following metric on it: given two vectors $\mathbf{a}, \mathbf{b}$ in Σ^n, let us define the distance between them as the number of components in which they differ; that is, $d(\mathbf{a}, \mathbf{b}) = |\{1 \le i \le n, a_i \ne b_i\}|$ We define the t-disc of $\mathbf{a}$ as the set of points at distance $\le t$ from $\mathbf{a}$ i.e. $disc_t(\mathbf{a}) = \{\mathbf{b} \in \Sigma^n : d(\mathbf{a}, \mathbf{b}) \le t\}$

We will define again VSS as a pair of protocols, called `Share-Verify` and `Recover`, that compute, respectively, two functions, SHAR and REC, satisfying the following properties. SHAR is the function we use to share the secret among the players. It is defined on the entire space for the $n-1$ players (their private input does not matter in this phase) and on two finite special sets R and S for the dealer. S is the space of possible secrets while R is a set of random strings. We will ask even after seeing any l shares ($l \le t$) all secrets are equally likely to generate those shares. We call this property *t-uniformity* (see 2 below). Similarly REC is the function we use to reconstruct the secret. We will run it on the output of the previous phase. What we want is that we will be able to do so even if up to t components of the output of the sharing process are arbitrarily changed. We call this property *t-robustness* of the function REC (see 3 below).

Definition: Two functions SHAR and REC are a *sharing-reconstructing pair* with parameter t if they have the following properties:

1. (*Domain.*)
$$\text{SHAR} : \Sigma^{n-1} \times (R \times S) \to \Sigma^n$$
$$\text{REC} : \Sigma^n \to \Sigma^n$$
2. (*t-uniformity.*) $\forall l \le t$ there exists an integer n_l such that $\forall\ s_{i_1}, \ldots, s_{i_l} \in \Sigma$ and $\forall\ v_1, \ldots, v_{n-1} \in \Sigma$, $\forall\ s \in S$, and $\forall \mathbf{x} \in \Sigma^n$ such that $\forall j \in [1, l]\ x_{i_j} = s_{i_j}$, there exist exactly n_l values $r_1, \ldots, r_{n_l} \in R$ such that for $i = 1, \ldots, n_l$,
$$\text{SHAR}(v_1, \ldots, v_{n-1}, r_i \circ s) = \mathbf{x}$$

3. (t-*robustness.*) $\forall\ v_1, \ldots, v_{n-1} \in \Sigma$, $\forall\ s \in S$, $\forall\ r \in R$,
   if $\mathbf{x} \in disc_t(\textsc{shar}(v_1, \ldots v_{n-1}, r \circ s))$, then

$$\textsc{rec}(\mathbf{x}) = (s, s, \ldots, s)$$

A VSS protocol will be composed by two protocols that *securely* evaluate these two functions; the second being evaluated over the output of the first.

Definition 4. A VSS protocol of fault-tolerance t is a pair of protocols (`Share-Verify`, `Recover`) such that

– `Share-Verify` securely evaluates the function $\mathbf{y} = \textsc{shar}(x_1, \ldots, x_{n-1}, r \circ s)$,
– `Recover` securely evaluates the function $\textsc{rec}(\mathbf{y})$, and
– $\textsc{shar}$ and $\textsc{rec}$ are a sharing-reconstructing pair with parameter t.

Remarks: Though the above definition may appear "tailored on some specific VSS protocols," in the final paper we shall argue that it does not loose any generality. Also, as we shall see below, by demanding that both components (and particularly the second one) of a share-reconstructing pair be securely evaluated, we are putting an unusually strong requirement on a VSS protocol. But it is exactly this requirement that will guarantee the desired reducibility property.

4 Comparison with previous definitions of VSS

Let us compare now Definition 4 and Definition 1, our token example of prior VSS definitions. To begin with, there is a minor syntactical difference between the two definitions: according to Definition 1 when good players find out the dealer is bad they just stop playing and output $verification = no$. In our new definition instead the computation goes on, no matter what. This discrepancy can be eliminated by having protocols in the first definition agree on a default value when the dealer is clearly bad and protocols in the second definition always output $verification = yes$ at the end of `Share-Verify` (since we are dealing with a secure function evaluation, we are guaranteed that all good players will output a common value). With these minor changes we can prove the following (the proof can be found in the appendix):

Theorem 5. *If P is a VSS protocol of fault-tolerance t satisfying Definition 4, then P is also a protocol of fault-tolerance t satisfying Definition 1.*

Now the natural question to ask is: Are Definitions 3 and 1 equivalent? That is, if a given VSS protocol P' satisfies Definition 1, does it also satisfy Definition 3? The answer to this important question, provided by the following Theorem 3, is NO. And it better be that way if we want to preserve reducibility of VSS protocols. Indeed as we will see later the formal specifications of Definition 1 are not sufficient to guarantee composition of VSS protocols inside larger protocols.

Theorem 6. *Definition 4 is strictly stronger than Definition 1, that is, there are VSS protocols satisfying Definition 1, but not Definition 4.*

The proof of this theorem (see the appendix for a detailed proof) is based on the fact that usually VSS protocols (consider for example the one in [1]) reconstruct the secret by having each player distributing his own share to all other players. This does satisfy

Definition 1 since there we do not put any requirement on the secrecy of the shares. But this does not satisfy Definition 4 since doing so we do not compute *securely* the function REC. The problem is that the players reveal too much information about their own input share during the protocol. In other words we want that, when we compute $\text{REC}(\mathbf{y})$ over $\mathbf{y} = \text{SHAR}(v_1, \ldots, v_{n-1}, r \circ s)$, no knowledge about $\mathbf{y}$ should leak except the secret s (including no extra information about the secret s itself). The rationale for asking this is again the fact that we want our VSS protocols to be secure not just by themselves but when used inside subroutines of more complex protocols. Leaking knowledge about the shares (or extra knowledge about the secret) may create problems to the security of the overall protocol. Consider the following example.

Consider a VSS protocol P, satisfying Definition 1, in which the secret is a 3-colorable graph. During the `Recover` protocol the graph is reconstructed together with a 3-coloring of it kindly provided by the dealer. Notice that Definition 1 is not violated, but notice also that an adversary gains from the execution of such a protocol some knowledge about the secret she could not obtain by herself. This in turns means that there exists no simulator for this protocol and so that Definition 4 cannot be satisfied. And the serious problem with P is that, if used inside a larger protocol in which it is crucial that the knowledge of that particular 3-coloring stays hidden, P, though "secure" as a VSS protocol on its own, jeopardizes the security of the larger protocol.

This problem is solved by our definition substituting property 3 (unpredictability) with a stronger one based on zero-knowledge and simulatability, which is exactly what we do by asking for a secure computation of the functions SHAR and REC. In particular the secure computation of the function REC is the most important difference between the two definitions. In particular we want to stress the importance of maintaining the secrecy of the shares of each individual good player. In the appendix we will show that protecting the secrecy of the shares is necessary for some applications in which VSS is used as a subroutine inside some specific protocols.

Probably one of the reasons this point may appear somewhat moot is that in Shamir's secret sharing scheme [11] the shares consist of the value of a polynomial of degree t with free term s. For a t-adversary who corrupts exactly t players, knowing the secret is equivalent to knowing the shares of all players. In fact, knowing the t shares of the corrupted players and the secret at the end of `Recover`, she has $t+1$ points of the t-degree polynomial, and by evaluating the so inferred polynomial at the names of all good players, she easily computes all shares. However, we object that what happens to be true for the VSS protocols based on Shamir's scheme, may not be true for all VSS protocols [3]. And one should not "wire in" a general definition what happens to be true in a specific case. Moreover, even in Shamir-based VSS protocols, if the polynomial has degree strictly larger than the number of corrupted players, then it is no longer true that for the adversary knowledge of the secret is equivalent to knowledge of all shares. Indeed, as we will show later this is the crucial point in our application protocols described in the appendix. *It is thus needed that the knowledge gainable by an adversary at the end of a secure VSS protocol exactly coincides with the original secret whenever the dealer is honest.*

[3] For example consider Blakley secret sharing scheme [2] in which the secret is a point in a $t+1$-dimensional space and shares are random hyperplanes passing through that point.

5 A VSS protocol that satisfies our definition

In this section we will exhibit a VSS protocol satisfying our definition and of fault-tolerance $\frac{n}{3}-1$, by modifying an older protocol of Ben-Or, Goldwasser, and Wigderson [1]. The modification actually occurs only in the `Recover` part, and uses techniques also developed by [1], but within their "computational protocol" rather than in their VSS protocol. We have also found a protocol of fault-tolerance $\frac{n}{2}-1$ based on Tal Rabin's protocol [10], but we will not describe it here for space limitations.

Suppose we are dealing with a t-adversary $\mathcal{A}$. Let $n = 3t+4$ and $P_1, \ldots, P_{n-1}$, $P_n = D$ be the set of players, D being the dealer. We will make all our computations modulo a large prime $p > n$. It is known from the error-correcting codes theory that if we evaluate a polynomial f of degree $t+1$ over the $n-1$ different points i for $i = 1, \ldots, n-1$ then given the sequence $s_i = f(i)$ we can reconstruct the coefficients of the polynomial in polynomial time even if up to t elements in the sequence are arbitrarily changed. This is the well known Berlekamp-Welch variant of the Reed-Solomon error-correcting code. For details readers can refer to a standard text like [12]. Let K be a security parameter. With K/n we mean $\lceil \frac{K}{n} \rceil$.

The protocol appears in the boxes. The `Share-Verify` part is identical to the one in [1]. The `Recover` protocol is modified with respect to the one in [1] in order to make it a secure computation of the function REC. The basic idea is that each player will distribute his share to all the other players, but covering it up appropriately with some randomness so that no information about the share is revealed but the secret reconstruction process is not compromised.

Theorem 7. *The protocol* $P = (\mathtt{Share-Verify}, \mathtt{Recover})$ *is a VSS protocol according to Definition 4 with fault-tolerance* $\frac{n}{3}-1$

Remark: A completely error-free version of this protocol can be obtained as in [7] by running a different zero-knowledge proof that the shares lie on a single polynomial. The proof uses a bivariate polynomial and it is out of the scope of this paper. Details can be found in [1, 7]. Notice that this is the first time a formal proof of the security of the protocol of [1] appeared.

Remark: Notice how, assuming the dealer is honest, at the end of the `Recover` phase the adversary, even knowing the secret s and t shares of the corrupted players, knows nothing about the shares of the honest players (since the polynomial is of degree $t+1$ and to interpolate it $t+2$ points are needed).

Protocol `Share-Verify` **from [1]**

1. The dealer chooses a random polynomial $f_0(x)$ of degree $t+1$ with the only condition that $f_0(0) = s$ his secret. Then he sends to player P_i the share $s_i = f_0(i)$. Moreover he chooses $2K$ random polynomials $f_1, \ldots, f_{2K}$ of degree $t+1$ as well and sends to P_i the values $f_j(i)$ for each $j = 1, \ldots, 2K$.
2. Each player P_i broadcasts K/n random bits $\alpha_{(i-1)K/n+j}$ for $j = 1, \ldots, K/n$
3. The dealer broadcasts the polynomials $g_j = f_j + \alpha_j f_0$ for all $j = 1, \ldots, K$
4. Player P_i checks if the values he holds satisfy the polynomials broadcast by the dealer. If he finds an error he broadcasts a complaint. If more than t players complain then the dealer is faulty and all players assume the default zero value to be the dealer's secret.
5. If less than t players complained the dealer broadcasts the values he sent in the first round to the players who complained.
6. Each player P_i broadcasts K/n random bits $\beta_{(i-1)K/n+j}$ for $j = 1, \ldots, K/n$
7. The dealer broadcasts the polynomials $h_j = f_{K+j} + \beta_j f_0$ for all $j = 1, \ldots, K$
8. Player P_i checks if the values he holds and the values broadcast by the dealer in round 5 satisfy the polynomials broadcast by the dealer. If he finds an error he broadcasts a complaint. If more than t players complain then the dealer is faulty and all players assume the default zero value to be the dealer's secret.

Protocol `Recover` **(modified)**

1. Each player P_i chooses a random polynomial h_i of degree $t+1$ such that $h_i(0) = s_i$ his own input share. He sends to player P_j the value $h_i(j)$
2. Each player P_i chooses random polynomials $p_i(x), q_{i,1}(x), \ldots, q_{i,2K}(x)$ of degree $t+1$ and with free term 0. He sends to player P_j the values $p_i(j), q_{i,1}(j), \ldots, q_{i,2K}(j)$
3. Each player P_i broadcasts K random bits $\gamma_{l,(i-1)K/n+m}$ for $l = 1, \ldots, n$ and $m = 1, \ldots, K/n$
4. Each player P_i broadcasts the following polynomials $r_j = q_{i,j} + \gamma_{i,j} p_i$ for each $j = 1, \ldots, K$
5. Each player P_i checks that the information player P_l sent him in round 1 is consistent with what player P_l broadcast in round 3. If there is a mistake or P_l broadcast a polynomial with non-zero free term P_i broadcasts bad_l. If there are more than t players broadcasting bad_l, player P_l is disqualified and all the other players assume 0 to be P_l's share. Otherwise P_l broadcasts the information he sent in round 1 to the players who broadcast bad_l
6. Each player P_i broadcasts K random bits $\delta_{l,(i-1)K/n+m}$ for $l = 1, \ldots, n$ and $m = 1, \ldots, K/n$
7. Each player P_i broadcasts the following polynomials $r_j = q_{i,K+j} + \delta_{i,j} p_i$ for each $j = 1, \ldots, K$
8. Each player P_i checks that the information player P_l sent him in round 1 and broadcast in round 5 is consistent with the polynomials player P_l broadcast in round 7. If there is a mistake or P_l broadcast a polynomial with non-zero free term P_i broadcasts bad'_l. If more than t players broadcast bad'_l then P_l is bad and all players assume his share to be 0.
9. Each player P_i distributes to all other players the following value $s_i + p_1(i) + p_2(i) + \ldots + p_n(i)$ then interpolates the polynomial $F(x) = f_0(x) + p_1(x) + p_2(x) + \ldots + p_n(x)$ using the error correcting algorithm of Berlekamp and Welch. The secret will then be $s = F(0) = f(0)$.

6 Conclusion

In the past cryptographic schemes and protocols used to be considered secure until not broken. Due to the increasing use and importance of cryptography, this approach is no more acceptable. To call a protocol *secure* we need a proof of its security. This means that we need definitions and methods to be able to prove security.

Following this philosophy we have presented a new and stronger definition for one of the most important cryptographic protocols: Verifiable Secret Sharing. We argued that this definition is the correct one especially when VSS is to be used as a subprotocol inside larger protocols (which is probably the most common case for VSS). We also presented a protocol which provably satisfies our new definition. Finally some applications of this new protocol (and of our new definition of VSS) are described in the appendix.

References

1. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th ACM Symposium on Theory of Computing*, pages 1–10, 1988.
2. G.R. Blakley. Safeguarding cryptographic keys. In *National Computer Conference*, pages 313–317, 1979.
3. David Chaum, Claude Crepeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *20th ACM Symposium on Theory of Computing*, pages 11–19, 1988.
4. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th IEEE Symposium on Foundations of Computer Science*, pages 383–395, 1985.
5. Yvo Desmedt and Yair Frankel. Shared generation of authentication and signatures. In *CRYPTO'91*, Lecture Notes in Computer Science, pages 457–469. Springer-Verlag, 1991.
6. Yvo Desmedt, Yair Frankel, and Moti Yung. Multi-receiver/multi-sender network security: efficient authenticated multicast/feedback. In *INFOCOM*, pages 2045–2054, 1992.
7. Paul Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. In *20th ACM Symposium on Theory of Computing*, 1988.
8. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *19th ACM Symposium on Theory of Computing*, pages 218–229, 1987.
9. Silvio Micali and Philip Rogaway. Secure computation. In *CRYPTO'91*, Lecture Notes in Computer Science. Springer-Verlag, 1991. Current version available from the authors.
10. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM Symposium on Theory of Computing*, 1989.
11. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
12. W.Peterson and E.Weldon. *Error Correcting Codes*. MIT Press, second edition, 1972.

Appendix

This appendix is dedicated to the proofs of some of the statements in the the paper.

Proof of Theorem 5

Let P be a VSS protocol of fault-tolerance t satisfying Definition 4. Let **ic** be the initial configuration vector of the network.

P satisfies the Verifiability property of Definition 1. Indeed at the end of the `Share-Verify` phase, let $\sigma = I_n(ic_n)$ i.e. the value contributed by the dealer to the computation. This value is *fixed* at the end of `Share-Verify` since it is a function of the traffic of the dealer. Moreover because of the t-robustness of the function REC we have the same value σ will be output by all good players at the end of the `Recover` part. Indeed because of the t-robustness property it does not matter that t bad players may change their input before computing the function REC. Finally, if the dealer is good then $\sigma = I_n(ic_n)$ is equal to the secret s.

P also satisfies the Unpredictability property of Definition 1. First notice that the t-uniformity property implies that the output of the function SHAR is composed of t-wise independent uniformly distributed random variables and so it is impossible (in an information-theoretic sense) to predict the secret better than at random for any algorithm that has knowledge of only $l \leq t$ components of such output. However a t-adversary has not only that knowledge but she also has a view of the entire protocol (i.e. traffic of bad players and messages broadcast by good players). But here is where secure computation comes to our rescue. Because of the security of the evaluation of the function SHAR the adversary can create the entire view by herself using the simulator, and so basically the other information is irrelevant. More precisely let's assume for the sake of contradiction that protocol P does not satisfy the Unpredictability condition. This means that there is an adversary $\mathcal{A}$ that does not corrupt the dealer and such that $Prob[\mathcal{A}$ guesses $s] > \frac{1}{m}$ where $m = |S|$ is the cardinality of the space of possible secrets. We will exhibit an algorithm $\mathcal{G}$ (for guesser) that guesses the secret with the same probability but having access only to t components of the output of the function SHAR. Since `Share-Verify` is a protocol that securely evaluates the function SHAR it must have a simulator Sim. $\mathcal{G}$ runs Sim with the adversary $\mathcal{A}$ and uses $\mathcal{A}$'s guess for the secret as his guess. Since the simulated view of the protocol is indistinguishable from the real one, $\mathcal{A}$ will guess the correct secret with probability bigger than $\frac{1}{m}$. Notice that by running the simulator $\mathcal{G}$ will only know at most t components of the output of the function SHAR, the ones which will be answered by the oracle in response to request of corruption by $\mathcal{A}$. So we have a contradiction since we found an algorithm that predicts the secret better than at random with knowledge of only t components of the output of SHAR, which contradicts the t-uniformity property. □

Proof of Theorem 6

The proof of this theorem is based on the fact that usually VSS protocols (consider for example the one in [1]) reconstruct the secret by having each player distributing his own share to all other players. This does satisfy Definition 1 since there we do not put any requirement on the secrecy of the shares. But, as the following lemma shows, this does not satisfy Definition 4 since doing so we do not compute *securely* the function REC.

Lemma 8. *The protocol P in which at round 1 every player broadcasts or distributes his own share is not a secure computation of the function* REC.

Proof. P is a 1-round protocol. Consider the following t-adversary $\mathcal{A}_1$. $\mathcal{A}_1$ corrupts t players (say $P_1, \ldots, P_t$) right at the beginning of the protocol. So in the simulated execution the simulator Sim will receive from the oracle their inputs (shares) $x_1, \ldots, x_t$. Now Sim has to simulate the broadcast messages of the good players. No matter what *any* Sim does, there are only two possibilities:

- either the shares broadcast by Sim do not interpolate a secret
- or they do interpolate a secret s', but since Sim has no knowledge at this point of what the "true" secret s is, only with probability $\frac{1}{|S|}$ $s = s'$

In both cases however the simulated execution is distinguishable from the real one to $\mathcal{A}_1$ and so protocol P is not secure according to definition 2.

The above proof is sufficient to show that protocol P is not secure. However let us describe an alternative proof of this statement, based on a different adversary. This will allow us to exemplify a different problem with protocol P. This will help the reader understand better the modification we will do to the `Recover` protocol in order to achieve security for VSS protocols.

Proof. (alternative proof of Lemma 8)
Consider the following t-adversary $\mathcal{A}_2$. $\mathcal{A}_2$ corrupts $t-1$ players (say $P_1, \ldots, P_{t-1}$) right at the beginning of the protocol. So in the simulated execution the simulator Sim will receive from the oracle their inputs (shares) $x_1, \ldots, x_{t-1}$. Now Sim has to simulate the broadcast messages of the good players. Sim does so and broadcasts $x'_t, \ldots, x'_n$. As before no matter what *any* Sim does, there are only two possibilities:

- either the shares broadcast by Sim do not interpolate a secret (in this case, as above, the simulation is already a bad one)
- or they do interpolate a secret s',

At this point $\mathcal{A}_2$ corrupts her last player (say P_t) and so gets from the oracle the true pair (x_t, s) and only with probability $\leq \frac{1}{|S|}$ we have that $x'_t = x_t$ and $s = s'$ and so the simulated execution will not succeed in convincing the adversary that she is talking with the real network, hence P is not secure according to definition 2.

Proof of Theorem 7

Let SHAR be the following function:

$$\text{SHAR}(v_1, v_2, \ldots, v_{n-1}, r \circ s) = (s_1, s_2, \ldots, s_{n-1}, \epsilon)$$

with $s_i = f_0(i)$ where $f_0(x) = s + a_1 x + \ldots + a_{t+1} x^{t+1}$ and $r = a_1 \circ \ldots \circ a_d$ (i.e. the polynomial is created using the coin tosses r of the dealer). Then we can state that

Lemma 9. *Protocol* `Share-Verify` *securely evaluates the function* SHAR *according to Definition 2.*

Proof. If we look back at Definition 2 we see that we have to check that both conditions, correctness and privacy, are satisfied in the *blended* way they interact in the simulation process through the common input and output functions.

So first of all let's define these functions for our protocol. Remember that with t_i we define the traffic of player P_i. Clearly for all players P_i $i \leq n$ the input function always returns the empty string, $I_i(t_i) = \epsilon$, since the players do not contribute any input during the computation of the function SHAR. For the dealer, $D = P_{n+1}$, the input function is a little bit more complicated. Let us denote with m_i the message the dealer broadcast to player P_i in round 5 if P_i complained in round 4, or the message the dealer sent to player P_i in round 1 if P_i did not complain. Then $I_D(t_D) = f(0)$ where $f = BW(m_1, \ldots, m_n)$ is the $t+1$-degree polynomial resulting from the Berlekamp-Welch interpolation of the m_i's. The output function is simpler: $O_i(t_i) = m_i$ (where $m_D = \epsilon$). Now we can check both conditions.

Correctness: First we have to prove that for all good players P_i, $I_i(t_i)$ is equal to the correct input. In our specific case this has to be checked only for the dealer. If the dealer is good, $m_i = f(i)$ where f is a $t+1$-degree polynomial with free term s the secret. So

$I_D(t_D) = s$ if the dealer is good. The second correctness condition is that with high probability $\mathbf{O}(\mathbf{t}) = \text{SHAR}(\mathbf{I}(\mathbf{t}))$. In our case this means that with high probability the values m_i held by good players must be on a single polynomial of degree $t+1$. This is true with probability $\geq 2^{-\frac{2K}{3}}$ since at least $\frac{2K}{3}$ bits are chosen truly randomly by good players in rounds (2) and (6). Each bit represents a "question" that a bad dealer who distributed bad shares will be able to answer correctly in the following round only with probability $\frac{1}{2}$ (i.e. if he predicted the bit correctly when he distributed the shares). Hence the bound on the probability of error.

Privacy: We have to exhibit a simulator for the protocol. We distinguish 2 cases:

Case A: The dealer is corrupted before round 1. Then the simulator will just follow the instructions of the players, with the only exception that it will turn them over to the adversary in case of corruption. Since the players do not contribute any input to the computation this will reduce the simulated execution to one of VSS with a bad dealer. So the simulation will be indistinguishable to the eyes of the adversary.

Case B: The dealer is not corrupted before round 1. Then the simulator in round 1 will just create a random "fake" secret s' and will share it to the players according to the protocol instructions with a polynomial f'. If the dealer is not corrupted at all during the protocol then everything will run smoothly since to the eyes of the adversary the execution will look like an ordinary VSS with a good dealer (again this is true because the players do not contribute any input to the computation). If the dealer is corrupted after round 1 however the adversary and the simulator will get from the oracle the true input s of the dealer. At that point the simulator turns over the control of the dealer to the adversary, but changes the polynomial used to share the secret to a new polynomial f" such that f"$(0) = s$ and f"$(i) = f'(i)$ for all players P_i that were corrupted so far by the adversary. The simulator changes accordingly the random polynomials f_i used for the zero-knowledge proof to make them consistent with whatever has been broadcast so far. The simulator can always do this since the adversary has at most t points of a $t+1$-degree polynomial. For the rest of the simulation the simulator will use the polynomial f" for the computation of the good players still under his control. We claim that this execution is indistinguishable to the adversary from a real one. This is so because the only thing different from a true execution is the fact that the shares the adversary gets before corrupting the dealer are created using a different polynomial than the real one, but thanks to the properties of polynomials this is not a problem for the simulator once the dealer is corrupted.

Let REC be the function

$$\text{REC}(s_1, \ldots, s_{n-1}, \epsilon) = (s, \ldots, s, s)$$

where s is the result of the Berlekamp-Welch "interpolation" of the s_i.

Lemma 10. *Protocol* `Recover` *securely evaluates the function* REC *according to Definition 2.*

Remark: Before embarking in the formal proof of this lemma let us give some intuition of why this is true. For example notice how the adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ that we described in the proof of Lemma 8 are helpless with the new protocol `Recover-A`. We added round 1 so that the simulator can learn the true secret *before* the shares have been given out publicly and this takes care of an adversarial attack like $\mathcal{A}_1$. Moreover we added the "masking" polynomials p_i so that the players reveal shares of a random polynomial F whose only property is that $F(0) = s$, so while reconstructing the secret no information is revealed about the true input shares; this solves the problem raised by adversary $\mathcal{A}_2$.

Proof. As in the proof of Lemma 9 we start by defining the input and output functions of our protocol. The input function I_i of player P_i is defined as follow: let $m_{i,j}$ be the message P_i sends to player P_j at round 1; $I_i(t_i) = h_i(0)$ where $h_i = BW(m_{i,1}, \ldots, m_{i,n})$ is the $t+1$-degree polynomial resulting from the Berlekamp-Welch interpolation of the $m_{i,j}$'s (if there is no such polynomial then assume $I_i(t_i) = 0$). The output function is the following: let M_i be the message broadcast by player P_i at round 9; $O_i(t_i) = F(0) = s$ where $F = BW(M_1, \ldots, M_n)$ is the $t+1$-degree polynomial resulting from the Berlekamp-Welch interpolation of the M_i's.

Correctness: It is clear that for all good players $I_i(t_i) = s_i$ the correct input share. Then we have to check that at least with high probability $\mathbf{O}(\mathbf{t}) = \text{REC}(\mathbf{I}(\mathbf{t}))$. In our case this means to prove that

$$\text{REC}(M_1, \ldots, M_n, \epsilon) = \text{REC}(s_1, \ldots, s_n, \epsilon)$$

Now this equation is not satisfied if one of the following things happens:

- either a bad player P_l succeeds in sharing random "garbage" instead of the values $p_l(j)$ in round 2 (in this case the M_i's will not interpolate a polynomial)
- or P_l does distribute $p_l(j)$ in round 2 but manages to use a polynomial with free term different than zero (in this case the M_i's will reconstruct a different secret)

Since the sharing process is exactly identical to the one of the protocol `Share-Verify`, we already know that P_l succeeds in any of the two cases only with probability $2^{-\frac{2K}{3}}$. So since there are at most $\frac{n}{3}$ bad players, the probability that the protocol computes an incorrect output is at most $\frac{n}{3}2^{-\frac{2K}{3}}$ which for K large enough is exponentially small.

Privacy: We have to exhibit a simulator and then prove that the simulation is indistinguishable from the true network execution. Consider the following simulator Sim_R:

1 At round 1, Sim_R simulates player P_i by choosing a random polynomial h_i' of degree $t+1$ and sending $h_i'(j)$ to P_j. At this point the simulator is allowed to receive from the oracle the output of the function, so Sim_R will learn the true secret s. If some player P_l is corrupted by the adversary $\mathcal{A}$ at the end of this round (or in the following rounds), then both Sim_R and $\mathcal{A}$ learn the true share s_l and Sim_R has to change the polynomial h_l' accordingly so that $h_l'(0) = s_l$ but without changing its value on points already known to the adversary. Sim_R can always do this because the adversary has at most t points of a $t+1$ degree polynomial.

2-8 During rounds 2 to 8 the simulator just follows plainly the instructions of the players. Since what players do in these rounds is completely random and not related to their inputs, Sim_R will always be able to create an indistinguishable view.

9 Finally at round 9, Sim_R chooses a polynomial g of degree $t+1$ such that $g(0) = s$ and then for each player P_i Sim_R broadcasts $g(i) + p_1(i) + \ldots + p_n(i)$ where p_j is the polynomial distributed by player P_j during rounds 2-8 of the simulation. The Reed-Solomon interpolation of these values will give as result s. If a player P_l is corrupted at the end of this round, then both Sim_R and $\mathcal{A}$ will learn from the oracle the true input share s_l. If $s_l \neq g(l)$ then Sim_R just changes the value of p_l at the point l so to make the entire sum consistent with what broadcast.

The simulation is indistinguishable from a real execution to the eyes of the adversary. In fact as we already said, in round 2-8 all messages are random and unrelated to the input so the simulator can easily play the role of the good players. In round 1 the adversary sees at most only t shares of the real input of a good player. Because of the property of Shamir secret sharing scheme, these shares are completely random and so can be simulated even with no knowledge of the real input (as in the case of the simulator). In round 9 the real share is broadcast "hidden" by some random "garbage", this will allow the simulator to broadcast the message of a good player with the right distribution even without knowing the real input.

Applications

Unfortunately there is no space in this extended abstract for a detailed description of the applications. However let us briefly sketch some of them.

A PROOF FOR THE BGW PROTOCOL: In the final paper we will describe what is probably the nicest feature of our new definition and new protocols for VSS. We will present a very simple and "modular" proof for the theorem of Ben-Or, Goldwasser and Wigderson that any function can be computed securely with fault-tolerance of $\frac{n}{3}$. This result first appeared in a STOC abstract without a formal proof [1]

They claim that it is enough to prove that it is possible to compute securely addition and multiplication, since any function F can indeed be reduced to an arithmetic circuit whose nodes are indeed just addition and multiplication. By repeatingly using these 2 protocols we can then compute securely the entire function. Notice however that this line of reasoning gives for granted the reducibility property for secure protocols. However at that time a satisfactory definition of security had not been presented and the reducibilty theorem was generally considered true but never proven.

However we stand on the privileged position of having had these tasks completed for us by [9]. This will allow us to present a simple and modular proof for the theorem of [1]. Notice that in order to use the reducibility theorem we *need* to use our new notion and protocols for VSS. This does not mean that the theorem of [1] is wrong as it is, but that in order to have our simpler and modular proof their VSS protocol is not appropriate.

SHARED AUTHENTICATIONS AND SIGNATURES: A first idea would be to use these shares for *authentication* purposes. More precisely after being shared among the players, the secret is given to some *authenticators*. At a certain point if a subset of the players (of opportune size) wants to prove their identity to or sign a message for an authenticator they just recover the secret together with her and if the reconstruction succeeds then the authenticator knows she is dealing with the right people. However since the shares are never revealed the authenticator (or any of the players) will never be able to prove herself as someone else.

The idea of shared authentication and signatures appears in [5, 6]; we will briefly describe their scheme here. All operations (as usual) are modulo a big public prime p. A trusted key distribution center generates two random polynomials $Q_0(x)$ and $Q_1(x)$ of degree t and shares them among the players, i.e. player P_i receives $Q_0(i)$ and $Q_1(i)$. The authenticators receive $Q_0(0)$ and $Q_1(0)$. Later suppose players $P_{i_1}, \ldots, P_{i_s}$ $(s \geq t)$ want to sign a message M for the authenticator. P_{i_j} broadcasts the message $Q_0(i_j) + MQ_1(i_j)$. The authenticator interpolates the broadcast shares into a polynomial F and checks that $F(0) = Q_0(0) + MQ_1(0)$. This protocol has two problems:
(1) If a player wants to jam the signature process all he has to do is to broadcast garbage (they solve it assuming players to be honest).
(2) It is a one-time scheme since after signing two different messages M_1 and M_2 the secret key of player P_i is easily computable.

Using our VSS `Recover` protocol to reconstruct the secret $Q_0(0) + MQ_1(0)$ we will solve both problems at once. Indeed bad shares contributed by players will be detected and eliminated. Moreover the share broadcast by player P_i betrays nothing of the true secret $(Q_0(i), Q_1(i))$ and so even after numerous signatures no information leaks about it; so we can do without the trusted combiner.

Finally notice that since usually the key management center is trusted, we do not need the verification process during the `Share-Verify` part making that part much simpler.