Verifiable Secret Sharing as Secure Computation

Rosario Gennaro Silvio Micali

Laboratory for Computer Science
Massachusetts Institute of Technology
rosario,silvio@theory.lcs.mit.edu

Abstract

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.

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.

THE INTUITIVE NOTION OF A VSS. As first introduced by Chor, Goldwasser, Micali and Awerbuch in [3], 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 (again, no matter what the bad players might do).

PRIOR WORK. Several definitions and protocols for VSS have been proposed in the past ten years (E.g., [3, 1, 2, 4, 11].) 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." 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. They are not, however, adequate in more general scenarios: when VSS is used as a tool towards other ends, that is, when it is used as a *sub-protocol* within a larger protocol. Indeed, and unfortunately, 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. (For instance, [4] used their VSS as a tool to reach Byzantine agreement, and thus had to argue that their overall protocol was secure "from scratch" rather than in a "modular way." Of course, such proofs from scratch tend to be overly long and complex.)

OUR WORK. In this paper we put forward a definition of VSS that guarantees *reducibility;* that is, security even when VSS is used as a sub-routine in an otherwise secure protocol. A notion of security that guarantees reducibility has been presented by Micali and Rogaway [9], but for the problem of *function evaluation.* We thus wish to 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 based on secure function evaluation
   (This will guarantee the reducibility that characterizes the latter notion.)
2. We compare our new notion with the previously proposed ones, and show that it is *strictly and inherently stronger.*

(Indeed, though sometimes protocols proved to satisfy weaker properties also satisfy stronger ones, we shall also show that none of the previous VSS protocols can satisfy our new notion.)

3. We modify an earlier VSS protocol of [1] and show that it is secure according to our notion.

2 Prior work

In order to focus on the difficulties that are proper of VSS, in this extended abstract 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 the messages received by the bad players, 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 [4] 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 [4], 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. [1]

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 [6], [9] call a protocol for evaluating f secure if it 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 $\vec{ic}$, whose ith component, $ic_i = (x_i, r_i)$ consists of the private input and the random tape of player P_i.

[1] 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.

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* $\vec{I} = (I_1, \ldots, I_n)$ 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." $\vec{I}(\vec{ic})$ will denote the vector of those values when P is run on initial configuration $\vec{ic}$.

A *local output function* $\vec{O} = (O_1, \ldots, O_n)$ 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*, $VIEW^{\mathcal{A}}_{Network}$, 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 $\vec{I}$ 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.
- 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').

Definition 2 *(secure function evaluation)*
Let f be a vector-valued function, P a protocol, Sim a simulator, and $\vec{I}$ and $\vec{O}$ local input and output functions. We say that P securely evaluates the function f if

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

  (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 $\vec{ic}$, if $VIEW^{\mathcal{A}}_{Sim}$ is the adversary view of the simulated execution of the protocol, we have that

$$VIEW^{\mathcal{A}}_{Network} \approx VIEW^{\mathcal{A}}_{Sim}$$

(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 [9] definition allows one to prove formally many desirable properties of secure protocols, the most interesting for us being *reducibility*:

Theorem 1 ([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 evaluation 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 (80-page!) 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 $\Sigma = \{0,1\}^*$. Consider the vector space Σ^n and the following metric on it: given two vectors $\vec{a}, \vec{b}$ in Σ^n, let us define the distance between them as the number of components in which they differ; that is,

$$d(\vec{a}, \vec{b}) = |\{1 \leq i \leq n, a_i \neq b_i\}|$$

We define the t-disc of $\vec{a}$ as the set of points at distance $\leq t$ from $\vec{a}$ i.e.

$$disc_t(\vec{a}) = \{\vec{b} \in \Sigma^n : d(\vec{a}, \vec{b}) \leq t\}$$

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 ($l \leq t$) 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: We say that two functions SHAR and REC are a *sharing-reconstructing pair* with parameter t if they have the following properties:

1. (*Domain.*)
$$\text{SHAR} : \Sigma^{n-1} \times (R \times S) \rightarrow \Sigma^n$$
and
$$\text{REC} : \Sigma^n \rightarrow \Sigma^n$$

2. (*t-uniformity.*) $\forall l \leq t$ 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 $\forall \vec{x} \in \Sigma^n$ 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 $i = 1, \ldots, n_l$,
$$\text{SHAR}(v_1, \ldots, v_{n-1}, r_i \circ s) = \vec{x}$$

3. (*t-robustness.*) $\forall\, v_1, \ldots, v_{n-1} \in \Sigma, \forall\, s \in S, \forall\, r \in R$, if $\vec{x} \in disc_t(\text{SHAR}(v_1, \ldots v_{n-1}, r \circ s))$, then
$$\text{REC}(\vec{x}) = (s, s, \ldots, s)$$

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[2].

Definition 3 *A VSS protocol of fault-tolerance t is a pair of protocols (*`Share-Verify`, `Recover`*) such that*

- `Share-Verify` *securely evaluates the function* $\vec{y} = \text{SHAR}(x_1, \ldots, x_{n-1}, r \circ s)$,
- `Recover` *securely evaluates the function* $\text{REC}(\vec{y})$, *and*
- SHAR *and* 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.

[2] In [4] they use the terminology *sequence protocols* for this kind of interaction between two protocols

4 Comparison with previous definitions of VSS

Let us compare now Definition 3 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 funciton evaluation, we are guaranteed that all good players will output a common value).

With these minor changes we can prove the following:

Theorem 2 *If P is a VSS protocol of fault-tolerance t satisfying Definition 3, then P is also a protocol of fault-tolerance t satisfying Definition 1.*

Sketch of Proof First, P satisfies the Verifiability property of Definition 1. Indeed because of the t-robustness of the function REC we have that at the end of the phase `Share-Verify` a value σ has been fixed and all the good players will output this value at the end of the `Recover` part. This is the value that can be obtained by applying the function REC to the output of the function SHAR. Because of the t-robustness property it does not matter that t bad players may change their input before computing REC. Moreover if the dealer is good this value σ is equal to the secret s.

Second, P also satisfies the Unpredictability property of Definition 1. Notice that because of the t-uniformity property 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 the output of the function SHAR. Any t-adversary has that knowledge but she also has a view of the entire protocol. But here is where the 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. So it's impossible for any t-adversary to predict the secret better than at random.

Details of the proof will be presented in the final paper. ■

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.

Theorem 3 *Definition 3 is stricly stronger than Definition 1, that is, there are VSS protocols satisfying Definition 1, but not Definition 3.*

We will prove this theorem formally in the final paper, but let us address here some of the intuition behind the proof. We start with an easier point.

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 3 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 with Definition 1 could be easily solved by substituting property 3 (unpredictability) with a stronger one based on zero-knowledge and simulatability of `Share-Verify`. In [4] they shortly address this point. But, still, this would not solve all the problems. Indeed, another important difference between our definition and the previous one is that we require the computation of the function REC to be *secure*, i.e. simulatable. VSS protocols usually perform the `Recover` phase by having each player distribute his share to the others. This is *not* simulatable.

Lemma 1 *Distributing the shares is not a secure computation of the function* REC.

In other words we want that, when we compute $\text{REC}(\vec{y})$ over $\vec{y} = \text{SHAR}(v_1, \ldots, v_{n-1}, r \circ s)$, no knowledge about $\vec{y}$ should leak except the secret s. 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 may create problems to the security of the overall protocol.

Probably one of the reasons this point was missed before was that in Shamir's secret sharing scheme 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 infered 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. 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 bigger than the number of corrupted players, then it is no longer true that knowledge of the secret is equivalent to knowledge of all shares. (In fact, one may even use such a protocol both for verifiably secret sharing a given value and for, say, flipping a coin.) 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.

5 A VSS protocol that satisfies our definition

In the final paper, we shall demonstarte that Rabin's VSS protocol can be modified so as to yield a VSS protocol, secure in our sense and with fault-tolerance $n/2$. For the time being, we will be content of exhibiting a simpler VSS protocol secure in our sense and with fault-tolerance $n/3$, 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.

Let $n = 3t + 1$ 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 $p > 2^{2n}$. Let ω be a primitive n-th root of the unity in Z_p. It is known from the error-correcting codes theory that if we evaluate a polynomial f of degree t over the n different points ω^i for $i = 1, \ldots, n$ then given the sequence $s_i = f(\omega^i)$ then we can reconstruct the coefficients of the polynomial in polynomial time even if up to t elements in the sequence are arbitrarily changed. For details on this error-correcting encoding of a polynomial known as the Reed-Solomon code readers can refer to a standard text like [10]. Let m be a security parameter.

Protocol `Share-Verify` ([1]):

1. The dealer chooses a random polynomial $f_0(x)$ of degree t with the only condition that $f_0(0) = s$ his secret. Then he sends to player P_i the share $s_i = f_0(\omega^i)$. Moreover he chooses m random polynomials $f_1, \ldots, f_m$ of degree t as well and sends to P_i the values $f_j(\omega^i)$ for each $j = 1, \ldots, m$.

2. Each player P_i broadcasts a random value α_i

3. The dealer broadcasts the polynomials $g_i = \sum_{k=0}^{m} \alpha_i^k f_k$ for all $j = 1, \ldots, m$

4. Player P_i checks if the values he holds satisfy the polynomials broadcasted by the dealer. If he finds an error he broadcasts a complaint. If more than $t + 1$ 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 $t + 1$ players complained the dealer broadcasts the values he sent in the first round to the players who complained.

6. Each player P_i broadcasts a random value β_i

7. The dealer broadcasts the polynomials $h_i = \sum_{k=0}^{m} \beta_i^k f_k$ for all $j = 1, \ldots, m$

8. Player P_i checks if the values he holds satisfy the polynomials broadcasted by the dealer. If he finds an error he broadcasts a complaint. If more than $t + 1$ players complain then the dealer is faulty and all players assume the default zero value to be the dealer's secret.

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 $s_i = f_0(\omega^i)$ where $f_0(x) = s + a_1 x + \ldots + a_t x^t$ 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 2 *Protocol* `Share-Verify` *securely evaluates the function* SHAR *according to Definition 2.*

Proof To be presented in the full paper. (No proof of this protocol has yet appeared.) ■

The `Recover` protocol is modified with respect to the one in [1] in order to make it a secure computation of the function REC.

Protocol `Recover` (Modified):

1. Each player P_i chooses random polynomials $p_i(x), q_{i1}(x), \ldots, q_{im}(x)$ all with free term 0. He sends to player P_j the values $p_i(\omega^j), q_{i1}(\omega^j), \ldots, q_{im}(\omega^j)$
2. Each player P_i broadcasts nm random bits $\gamma_{k,l}^i$
3. Each player P_i broadcasts the following polynomials $r_j = q_{ij} + \gamma_{i,j}^{j \bmod n} p_i$ for each $j = 1, \ldots, m$
4. Each player P_i checks that the information player P_k sent him in round 1 is consistent with what player P_k broadcasted in round 3. If there is a mistake or P_k broadcasted a polynomial with non-zero free term broadcasts bad_k
5. If there are more than $t+1$ players broadcasting bad_k, player P_k is disqualified and all the other players assume 0 to be P_k's share.
6. Each player P_i distributes to all other players the following value $s_i + p_1(\omega^i) + p_2(\omega^i) + \ldots + p_n(\omega^i)$ 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 Solomon and Reed. The secret will then be $s = F(0) = f(0)$.

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 Solomon-Reed "interpolation" of the s_i.

Lemma 3 *Protocol* `Recover` *securely evaluates the function* REC *according to Definition 2.*

Proof Omitted. Will be presented in the full paper. ■

And so it follows that

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

Sketch of Proof Immediate from Lemmas 2 and 3 once we prove that SHAR and REC are a sharing-reconstructing pair of parameter $\frac{n}{3}$. But this is obvious from the properties of polynomials and of the Reed-Solomon encoding. Details in the final paper. ■

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 sub-protocol inside larger protocols (which is probably the most common case for VSS). We finally presented a protocol which provably satisfies our new definition.

References

[1] M.BenOr, S.Goldwasser, A.Wigderson, *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*, Proceedings of STOC 1988

[2] D.Chaum, C.Crepeau, I.Damgard, *Multiparty Unconditionally Secure Protocols*, Proceedings of STOC 1988, pp.11-19

[3] B.Chor, S.Goldwasser, S.Micali, B.Awerbuch, *Verifiable Secret Sharing and achieving simultaneity in the presence of faults*, Proceedings of FOCS 1985, pp. 383-395.

[4] P.Feldman, S.Micali, *An Optimal Probabilistic Protocol for synchronous Byzantine Agreement*, Proceedings of STOC 1988, final version in MIT-LCS TR425.b

[5] O.Goldreich, S.Micali, A.Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. of the ACM, Vol.38, No.1, pp. 691-729, 1991

[6] O.Goldreich, S.Micali, A.Wigderson, *How to play any mental game*, Proceedings of STOC 1987

[7] S.Goldwasser, S.Micali, *Probabilistic Encryption*, J. of Computer and System Sciences, Vol.18, No.1, pp.186-208

[8] S.Goldwasser, S.Micali, C.Rackoff, *The knowledge complexity of interactive proof systems*, in SIAM J. Comput. Vol.18 No.1, pp 186-208, 1989.

[9] S.Micali, P.Rogaway, *Secure Computation*, Proceedings of CRYPTO 1991, final version available from the authors.

[10] W.Peterson, E.Weldon, *Error Correcting Codes*, MIT Press 1972

[11] T.Rabin, M.BenOr *Verifiable Secret Sharing and multiparty protocols with honest majority*, Proceedings of STOC 1989

[12] A.Shamir, *How to share a secret*, Comm. of the ACM, 22(11), Nov. 1979, pp. 612-613