Universally Composable and Statistically Secure Verifiable Secret Sharing Scheme Based on Pre-Distributed Data

Rafael Dowsley[1], Jörn Müller-Quade[2], Akira Otsuka[3],
Goichiro Hanaoka[3], Hideki Imai[3], Anderson C. A. Nascimento[1]

[1] Department of Electrical Engineering, University of Brasilia
Campus Darcy Ribeiro, 70910-900, Brasilia, DF, Brazil.
Email: rafaeldowsley@redes.unb.br, andclay@ene.unb.br
[2] Universität Karlsruhe, Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5, 76128 Karlsruhe, Germany.
Email: muellerq@ira.uka.de
[3] National Institute of Advanced Industrial Science and Technology (AIST)
1-18-13, Sotokanda, Chyioda-ku, 101-0021, Tokyo, Japan.
Email: a-otsuka@aist.go.jp, hanaoka-goichiro@aist.go.jp, h-imai@aist.go.jp

Abstract. This paper presents a non-interactive verifiable secret sharing scheme (VSS) tolerating a dishonest majority based on data pre-distributed by a trusted authority. As an application of this VSS scheme we present very efficient unconditionally secure protocols for performing multiplication of shares based on pre-distributed data which generalize two-party computations based on linear pre-distributed bit commitments. The main results of this paper are a non-interactive VSS, a simplified multiplication protocol for shared values based on pre-distributed random products, and non-interactive zero knowledge proofs for arbitrary polynomial relations. The security of the schemes is proved using the UC framework.
Keywords: Verifiable Secret Sharing, Dishonest Majority, Pre-distributed Data, Unconditional Security, Universal Composable.

1 Introduction

This paper gives a protocol for information theoretically secure verifiable secret sharing (VSS) in the commodity based model which tolerates a dishonest majority. On the basis of pre-distributed data a dealer can share a secret such that all players are convinced that the shares they hold are valid, i. e., sets of players larger than a threshold t can reconstruct the shared secret. When dealing with an adversary which is able to corrupt a majority of the players, this requirement is slightly relaxed, since corrupted players are always able to abort the protocol.
As an application of this VSS scheme we present very efficient multiparty protocols for computing multiplication of shares in the commodity based model which can tolerate up to $t < n$ corrupted parties. If the number t is known the protocols can be chosen to be robust against $n - t$ players trying to abort the calculation. The protocols can be seen as a generalization of [21] to multiparty protocols. But the multiplication procedure used here is much simpler than the one in in [21] and compared to trivial extensions of [21] like pre-distributing two-party computations like oblivious transfer and then applying a general construction like [19] we save data. Due to the advantages of the commodity based model we can obtain a non-interactive VSS allowing non-interactive zero knowledge proofs.

VSS Tompa and Wolf [38] and McEliece and Swarte [31] were the first to study the problem of secret sharing in presence of a corrupted majority. They proposed solutions which work when the dealer is honest and corrupted players may attempt to cheat during the reconstruction of the secret. Later, Chor et al. [10] defined a complete notion of VSS, and gave a solution which was based on some intractability assumptions.

In Ben-Or et al. [5], an information theoretically secure VSS scheme was proposed which worked against any adversary which corrupts up to less than 1/3 of the players and the dealer. In [5], it was assumed that the players are connected by pairwise secure channels (the so-called secure channels model). As VSS, when implemented without a broadcast channel, implies Byzantine Agreement, the results of [29] show that the solution of [5] is optimal in the secure channels model.
Rabin and Ben-Or [36] were able to show that in the secure channels plus broadcast channel model unconditionally secure VSS is possible against any dishonest minority. In [11], Cramer et al. proposed a VSS scheme secure against an static adversary which can corrupt any dishonest minority by using a linear information checking protocol. The protocols of [11] and [36] are interactive.
While secrecy in the secure channels plus broadcast channel model can easily be maintained even against an adversary which corrupts a majority of players, the same cannot be said of the validity of the shares. Clearly, correct verifiability of the shares in the presence of a faulty majority cannot be achieved in the secure channels plus broadcast channel model without further assumptions [36].
Assuming that the discrete logarithm problem is intractable Feldman proposed a VSS scheme [15] where the verifiability of the shares is information theoretically secure but the secrecy of the secret is only computationally secure. Pedersen [34] proposed a "dual" of Feldman's scheme. Pedersen's scheme protects the secrecy of the secret unconditionally, while verifiability is protected only computationally under the assumption that the discrete logarithm problem is intractable.
In this paper, we introduce a VSS scheme based on pre-distributed data which is information theoretically secure against *dishonest majorities*, that is, both *the secrecy of the secret* and the *verifiability of the shares* are achieved independently of how much computational power is available to an adversary. Moreover, our solution is non-interactive.

Multiparty Computation As an application of our protocol, we provide a very simple and efficient information theoretically secure multiparty computation protocol for computing multiplication of shares based on pre-distributed data which is secure against dishonest majorities. this is known to imply the secure multiparty computation of any efficiently computable function. In [3, 4, 12] protocols for multiparty computations of multiplications secure against a faulty majority were proposed. In these papers, it was assumed that all the players were connected by oblivious transfer channels and a broadcast channel was available. Due to the use of pre-distributed data instead of oblivious transfer the protocols presented here are more efficient and do not need zero-knowledge proofs based on cut-and-choose arguments which increase the round complexity of protocols. Furthermore our protocol performs computations directly over a field $GF(q)$ and not only over binary fields.
Finally, it should be remarked that, if a protocol is secure against *any* dishonest majority, even a single player should be able to abort the computation as was pointed out in [12]. However, the protocols for secure computation proposed in [3, 4, 12, 19] can be aborted by a single player *even when the number of honest players is known to be much larger than one.* This is not the case with our protocol. Given that there are n players of which at most t are dishonest, then there exists a secure VSS protocol for which $n - t$ players are necessary to abort the execution of this protocol. This property also holds for the application to multiparty protocols.

Universal Composable Multiparty Computation Universally Composable (UC) Security [7] is a notion of security that holds even when the protocol is concurrently composed with an arbitrary set of protocols. This is a very desirable property for a protocol, since the protocols are normally executed concurrently to other tasks by each party. Canetti and Fischlin [8] showed that there exists functionalities that cannot be securely realized in the UC framework without any setup assumption in the case of an honest minority of parties. Some setup assumptions that allow UC secure multiparty computation are common reference string (CRS) model [8, 9, 14, 13], PKI model [1], random oracle model [23], signature cards [24], tamper-proof hardware [28, 33]

and noisy channels [26, 27]. The commodity-based model was used to obtain an UC-secure protocol to compute the inner product [25]. Prabhakaran and Sahai [35] introduced a model in which the environment, the adversary and the simulator are given oracle access to super-polynomial angels. In this model one can securely implement any multiparty functionality without setup assumptions.

1.1 Commodity Based Cryptography and Related Work

In [2] the commodity based cryptographic model was introduced on which the protocols presented here are based. In this model players buy cryptographic primitives from "off-line" servers. These primitives can be used later on to implement general cryptographic protocols. The commodity based model was inspired on the Internet architecture, which is usually based on the "client-server" paradigm. Once the primitives, or *commodities* as they are called by Beaver, are acquired, no further interaction between the server and the users is required. Therefore, the server need not know any secret values of the players.
In this contribution, we show that the use of off-lines servers provides very efficient and simple protocols for verifiable secret sharing and secure function evaluation over $GF(q)$ in the presence of a faulty majority.
Although this model was formalized just in [2], several independent works share the same flavor. We cite key-pre-distribution schemes [30], unconditionally secure bit commitments [37, 6] and unconditionally secure digital signature schemes [22].
The work which comes closest to the application of our VSS scheme to multiparty computations is [21]. There secure protocols for two-party computations in the commodity based model are proposed. Our protocol for multiparty secure computation can be understood as an extension of [21].

1.2 Our Contribution

In this Section we summarize our contribution. Due to the assumption that there is a trusted center which pre-distributes data during a setup phase, we could design a protocol for verifiable secret sharing which has the following interesting features.

- It is the first VSS protocol where the security of the secret and of the verifiability are achieved even against an all-mighty adversary which can corrupt any majority of the players and the dealer.
- It is non-interactive
- The verifiability of the protocol is not based on any cut-and-choose argument or expensive zero knowledge proofs.
- It is conceptually very simple and proven secure in the UC framework.

As an application of our VSS, we propose a protocol for secure multiparty computations of multiplications which also shows very interesting features (which to the best of our knowledge for the first time appear together in a single protocol): It is based on novel verifiable primitives in the commodity based model which allow two players to perform secure multiplication of shares over $GF(q)$; It is information theoretically secure against any adversary which can corrupt any majority of the players; and given that there are n players of which at most t are dishonest, $n-t$ players are necessary to abort an execution of the multiparty computation protocols.
Finally, to the best of our knowledge, our work is the first that formalizes the commodity based model in the UC framework.

2 The UC Framework

This section briefly review the main concepts of UC Framework, we refer the reader to [7] for a more detailed explanation of this framework. We also describe the ideal functionalities that we use in this work.

2.1 Overview

In the UC framework, the security of a protocol to carry out a task is established in three phases:

1. We formalize the process of executing a protocol in the presence of an adversary and an environment.
2. We formalize an ideal protocol for carrying out the task using a "trusted party". In the ideal protocol the trusted party captures the requirements of the desired task and the parties cannot communicate among themselves.
3. We prove that the real protocol emulates the ideal protocol.

The environment in the UC framework represents all activity external to the running protocol, so it provides inputs to the parties running the protocol and receives the outputs that the parties generate during the execution of the protocol. The environment tries to distinguish between attacks on real executions of the protocol and simulated attacks against the ideal functionality. If no environment can distinguish the two situations, the real protocol emulates the ideal functionality.
Proving that a protocol is secure in the UC framework provides the following benefits:

1. The ideal functionality describes intuitively the desired properties of the protocol.
2. The protocols are secure under composition.
3. The security is retained when the protocol is used as a sub-protocol to replace an ideal functionality that it emulates.

The ideal protocol. An ideal functionality $\mathcal{F}$ represents the desired properties of a given task. Conceptually, $\mathcal{F}$ is treated as a local subroutine by the several parties that use it, and so the communication between the parties and $\mathcal{F}$ is supposedly secure (i.e., messages are sent by input and output tapes).
The ideal protocol for an ideal functionality $\mathcal{F}$ involves an ideal protocol adversary $\mathcal{S}$, an environment $\mathcal{Z}$ on input z and a set of dummy parties that interacts as defined below. Whenever a dummy party is activated with input x, it writes x onto the input tape of $\mathcal{F}$. Whenever the dummy party is activated with value x on its subroutine output tape, it writes x on subroutine output tape of $\mathcal{Z}$. The ideal protocol adversary $\mathcal{S}$ has no access to the contents of messages sent between dummy parties and $\mathcal{F}$, and it should send corruption messages directly to $\mathcal{F}$ that is responsible for determining the effects of corrupting any dummy party. The ideal functionality receives messages from the dummy parties by reading its input tape and sends messages to them by writing to their subroutine output tape. In the ideal protocol there is no communication among the parties using the adversary to deliver the message. The environment machine, $\mathcal{Z}$, can set the inputs to the parties, and read their outputs, but it cannot see the communication with the ideal functionality.

The real protocol. In the real world, the protocol π is executed by parties $P_1, \ldots, P_n$ with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input z. $\mathcal{Z}$ can set the inputs for the parties and see their outputs, but it cannot see the communications among the parties.
The parties of π can invoke subroutines, pass inputs to them and receive outputs from them. They can also write messages on the incoming communication tape of the adversary. These messages may specify the identity of the final destination of the message. $\mathcal{A}$ can send messages to any party ($\mathcal{A}$ delivers the message). In addition, they may use the ideal functionalities that are provided to the real protocol.
$\mathcal{A}$ can communicate with $\mathcal{Z}$ and the ideal functionalities that are provided to the real protocol. $\mathcal{A}$ can also corrupt parties. After receiving a special message (Corrupt id) from the environment, the adversary corrupt a party by delivering the message (Corrupt). By the definition of the process of corrupting, the environment always knows which parties are corrupted.

The adversarial model. The parties have unique identities and are locally PPT. The network is asynchronous without guaranteed delivery of messages. The communication is authenticated and secure, and the parties have access to an authenticated broadcast channel. The secure channel between the parties and the authenticated broadcast channel can be pre-distributed by the trusted authority [2]. The adversary is static in corrupting parties, and is active in its control over corrupted parties. Finally, the adversary and the environment are allowed unbounded complexity. The simulator is allowed to be polynomial in the complexity of the adversary.

Realizing an ideal functionality. We say that a protocol π statistically UC-realizes an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-protocol adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input z, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π in the real-life process, or it is interacting with $\mathcal{S}$ and $\mathcal{F}$ in the ideal protocol. This means that, from the point of view of the environment, running protocol π is statistically indistinguishable of interacting with an ideal protocol for $\mathcal{F}$.

2.2 The Commodity Functionality

We describe below the functionality ($\mathcal{F}_{Commodity}$) that captures the Commodity Based Model [2] in which offline servers provides cryptographic primitives to the parties and do not engage in the rest of the protocol.

1. Upon receiving an input (DISTRIBUTE, *sid*, $P_{X_1,\ldots,X_n}$) where $P_{X_1,\ldots,X_n}$ is the specification of a probability distribution, verify that $sid = (sid_1, \ldots, sid_n, sid')$ where sid_i identifies a party; else halt. Next choose randomly $x_1, \ldots, x_n$ according to $P_{X_1,\ldots,X_n}$ and send a message (DISTRIBUTED, *sid*, $P_{X_1,\ldots,X_n}$, x_i) to the party specified by sid_i for $i = 1, \ldots, n$.

Since the specification of the probability distribution $P_{X_1,\ldots,X_n}$ and *sid* (that contains the identities of all participant parties in an ordered way) are specified in the outputs of the functionality, a corrupted party cannot gain advantage by choosing a probability distribution differing from the one specified in the protocol that utilizes this functionality.

2.3 The Multiplicative and Additive Verifiable Secret Sharing Functionality

We describe below a Multiplicative and Additive Verifiable Secret Sharing Functionality ($\mathcal{F}_{MA-VSS}$). In this functionality, various secrets can be shared among one set of parties (even secrets that have different dealers). Some specified subsets of the parties can together reconstruct the secrets. These subsets can also ask the functionality to multiply two secrets. In addition, a party can ask the functionality to compute a linear operation between two secrets. In linear and multiplication operations the secrets involved on the operation are not revealed to the parties.

1. When first activated, receive the list of the corrupted parties from the adversary. These parties will be added to the opening list of all secrets shared by this functionality.
2. Upon receiving an input (SHARE, *sid*, *vssid*, x_{vssid}) from some dealer D, verify that $sid = (\mathcal{Q}, sid')$ where $\mathcal{Q} = \{Q\}$ is an access structure, namely a collection of sets of identities $Q \subset \{0,1\}^*$, that $vssid = (D, vssid')$ and that there is no recorded secret for this *vssid*; else halt. Next, record $(sid, vssid, x_{vssid})$ and generate a public delayed output (SHARED, *sid*, *vssid*) to the parties in $\mathcal{R} = \bigcup_{Q \in \mathcal{Q}} Q$. Ignore subsequent (SHARE, $\overline{sid}$, ...) messages that have $\overline{sid}$ different from *sid*.
3. Upon receiving a message (OPEN, *sid*, *vssid*) from party P, add P to the (initially empty) list of parties that ask to open the secret and send a public delayed output (OPENING-ASKED, *sid*, *vssid*, P) to the parties in $\mathcal{R}$. If there is a set $Q \in \mathcal{Q}$ where all parties in Q have provided an (OPEN, *sid*, *vssid*) input, and there is a recorded shared value x_{vssid}, then generate a public delayed output (OPEN, *sid*, *vssid*, x_{vssid}) to the parties in $\mathcal{R}$.

4. Upon receiving an input (LINEAR-OPERATION, *sid*, $vssid_1$, λ, $vssid_2$) if there are recorded shared values corresponding to $vssid_1$ and $vssid_2$, compute $x_{vssid} = x_{vssid_1} + \lambda x_{vssid_2}$, record $(sid, vssid, x_{vssid})$ where $vssid = (\perp, vssid')$ is an identification to this new secret and generate a public delayed output (LINEAR-OPERATION-COMPUTED, *sid*, $vssid_1$, λ, $vssid_2$, *vssid*) to the parties in $\mathcal{R} = \bigcup_{Q \in \mathcal{Q}} Q$.
5. Upon receiving a message (MULTIPLY, *sid*, $vssid_1$, $vssid_2$) from party P, add P to the (initially empty) list of parties that ask to multiply the secrets and send a public delayed output (MULTIPLICATION-ASKED, *sid*, $vssid_1$, $vssid_2$, P) to the parties in $\mathcal{R}$. If there is a set $Q \in \mathcal{Q}$ where all parties in Q have provided an (MULTIPLY, *sid*, $vssid_1$, $vssid_2$) input and there are recorded shared values x_{vssid_1} and x_{vssid_2}, then compute $x_{vssid} = x_{vssid_1} \times x_{vssid_2}$, record $(sid, vssid, x_{vssid})$ where $vssid = (\perp, vssid')$ is an identification to this new secret and generate a public delayed output (MULTIPLICATION-COMPUTED, *sid*, $vssid_1$, $vssid_2$, *vssid*) to the parties in $\mathcal{R} = \bigcup_{Q \in \mathcal{Q}} Q$.

The linear and multiplicative operation among the shared secrets are defined inside the functionality in order to use the homomorphic properties of the instantiation of VSS to compute these operations.

3 The Verifiable Secret Sharing Protocol

In this section we describe first the VSS using the commodity functionality. Later we explain the protocol that can be used in order to prove polynomial relations among shared secrets using the VSS. In the next section we prove that together these protocols UC-realize the functionality $\mathcal{F}_{MA-VSS}$.

3.1 A VSS Protocol based on Pre-distributed Data

For a VSS scheme Π with access to a commodity functionality $\mathcal{F}_{Commodity}$, a set of dealers $\mathcal{D}$, and a finite set of players $\mathcal{P}$ our protocol works as follows:
We assume the existence of an authenticated broadcast functionality. Note that this assumption is made only to simplify the protocol presentation as a broadcast channel can as well be pre-distributed by the trusted center during a setup phase [2].
In the unconditionally secure protocol below the probability of cheating successfully equals the probability to successfully guess an element from the field over which computations are done. Hence we choose a prime power q and the field $GF(q)$ with q elements depending on the security parameter, e. g., $q = 2^k$. $\frac{t}{q}$ should be a negligible function of the security parameter.
The basic intuition behind the protocol is simple. The commodity functionality will share a random value with the players in a way that each player is committed to his share to each other player. When executing the Share algorithm, the dealer changes this random number, to which he is committed by the pre-distributed VSS, into a commitment to his secret.

Setup. When some player require the distribution of the commodities (specifying the identities of the dealer and the players that will share the secret), $\mathcal{F}_{Commodity}$ chooses for each player P_i a random verification key (a secret point) $w_i \neq 0$ from $GF(q)$ and output it to that player. It also randomly chooses a polynomial $f(x) \in GF(q)[x]$ such that

$$f(x) = \sum_{k=0}^{t} a_k x^k,$$

where each coefficient a_k is randomly and uniformly chosen from $GF(q)$, and t is the threshold of the secret sharing scheme. $\mathcal{F}_{Commodity}$ outputs the polynomial $f(x)$ to the dealer D.

In addition, for each player P_i[4] it chooses a random polynomial

$$v_i(x) = \sum_{k=0}^{t} b_{ik} x^k.$$

It outputs the polynomial $v_i(x)$ to P_i and the polynomial evaluations $v_i(w_j)$ to each player P_j such that $j \neq i$. It also broadcasts $v_i(0) + f(i)$. After delivering these private keys, $\mathcal{F}_{Commodity}$ does not engage in the rest of the protocol.
Therefore the probability distribution specification given to $\mathcal{F}_{Commodity}$ specifies that the functionality should: (1) choose randomly the verification points w_i of each player and output it to the party; (2) generate the random polynomial $f(x)$ and send it to the dealer; (3) choose the random verification polynomials $v_i(x)$ of each player, deliver the polynomial to such player and send to each other player the evaluation of the polynomial at the player's secret point; (4) for each player output $v_i(0) + f(i)$ for all players. Thus, during the setup phase, the dealer receives the polynomial $f(x)$ and each player P_i receives the secret point w_i, the polynomial $v_i(x)$, and the values $v_j(w_i), \forall j \neq i$ and $v_j(0) + f(j), \forall j$.
For the random "secret" $a = f(0)$, the share of a player is $f(i)$ and $v_i(x)$ is the verification polynomial for this share.

Share. In this stage of the protocol the shares of a random secret a will be changed to shares for a specific secret u. On input u for a secret, the dealer D computes a value c satisfying $c = u - a$, where a is computed as $a = f(0)$.
Then, D commits to his secret by broadcasting the value c. Each party send a message (SHARED, *sid*, *vssid*). The parties will add this value c to the reconstructed value after running the verification and reconstruction procedures.

Reconstruct. It is enough to show how a shared random secret a is reconstructed. The notation will therefore be as in the setup phase. Each player P_i broadcasts his polynomial $v_i(x)$ over the broadcast functionality.

Verify
On receiving $v'_j(x)$ from the player P_j over the broadcast functionality, each player P_i checks the share by checking the following equation:

$$ver_{ji} = \left\{ \begin{array}{c} accept \text{ if } v'_j(w_i) = v_j(w_i) \\ reject \text{ otherwise} \end{array} \right\}$$

If $ver_{ji} = accept$, P_i outputs (OPENING-ASKED, *sid*, *vssid*, P_j). If less than $n - t$ players are rejected by P_i, there will be a set of $t + 1$ valid shares $f(i_1), \ldots, f(i_t)$ in possession of P_i. Thus, the secret $a = f(0)$ can be reconstructed by Lagrange interpolation from $f(i_1), \ldots, f(i_t)$. Each player outputs (OPEN, *sid*, *vssid*, *a*). It is easy to see that (with overwhelming probability) all honest players will obtain the same result in the verification procedure.

3.2 Proving Polynomial Relations Among Shared Secrets

Note that in this protocol, for each player P_i the secret "check" information $w_i, 1 \leq i \leq n$ is never released, so it can be distributed only once for several protocols (even with different dealers). The check information can hence be safely reused within a bigger protocol and reduce communication from the trusted center to the players.
Another interesting fact about this VSS scheme is that, given that each player uses the same verification information w_i in all of its executions, it is linear, that is, the sum of two shares of

[4] The index i which is the "name" of a participant is here interpreted as a value of $GF(q)$. To have enough different names we need $q \geq n$. Due to security reasons, we must have $q \gg t$.

two secrets becomes a verifiable share of the sum of the secrets. We denote by $[a]_i$ the pair of the verification information and the share the player P_i holds from the secret a.
In this case all the commodities must be distributed jointly by $\mathcal{F}_{Commodity}$, since the polynomials are independent but the verification points w_i must be the same for all polynomials. So one player should send to $\mathcal{F}_{Commodity}$ the specification of the probability distribution instructing it to generate all the polynomials used in the linear operation and the verification points.

Proposition 1. *For two secrets a, b shared with the above VSS scheme using the same verification information w_i for each player P_i and a value $\lambda \in GF(q)$ it holds that $[a]_i + \lambda[b]_i = [a + \lambda b]_i$.*

When one party wants to execute a linear operation on the sharings, it broadcasts a message (LINEAR-OPERATION, *sid*, $vssid_a$, λ, $vssid_b$) to other parties. Each party calculates $[x]_i = [a]_i + \lambda[b]_i = [a + \lambda b]_i$ and then outputs (LINEAR-OPERATION-COMPUTED, *sid*, $vssid_a$, λ, $vssid_b$, $vssid_x$).
A linear VSS can be seen as a linear commitment to the shared secret. Using techniques from [21] it is possible to very efficiently prove polynomial relations among shared secrets.
For proving a linear relation among commitments one turns this relation into a set of linear functions which all must equal zero when evaluated on the committed values if and only if the relation holds. Proving that a given linear function evaluates to zero on committed values can be done by means of Proposition 1: using the linearity of the VSS scheme one computes from the given commitments a new commitment which represents the linear function evaluated on the given commitments and this new commitment is then opened to be zero.
To be able to prove arbitrary polynomial relations on committed values we will state a protocol that is based on a previous protocol from [21] which allows the parties to compute a new commitment which represents the product of two given commitments. This protocol can directly be applied to this linear secret sharing scheme. Note that the commodity functionality $\mathcal{F}_{Commodity}$ can pre-distribute shares to secrets which have a certain relation. The protocol for multiplication of secret is called a *distributed one time multiplication proof (DOTMP)* and consists of two phases: a pre-distribution phase where the commodity functionality shares additional values among the players (in this case all the commodities must also be distributed jointly by $\mathcal{F}_{Commodity}$ since the verification points w_i must be the same in all polynomials) and a non-interactive proof where the additional shared information is used to compute shares to the product of two shared values without reconstructing these.

Protocol DOTMP.

- Initialization: $\mathcal{F}_{Commodity}$ verifiably shares (with the players) three random numbers l, l' and l'', such that $l'' = ll'$. Thus, each player P_i receives $[l]_i, [l']_i$ and $[l'' = ll']_i$
- Multiplication: Each player P_i now holds shares to three random values l, l' and l'', such that $l'' = ll'$ as well as two shares $[a]_i$ and $[b]_i$ to the values a, b which are to be multiplied, to obtain a share $[ab]_i$ to ab each player P_i computes $[y]_i = [a]_i - [l]_i = [a - l]_i$ and $[y']_i = [b]_i - [l']_i = [b - l']_i$, and broadcasts the message (MULTIPLY, *sid*, $vssid_1$, $vssid_2$), $[y]_i$ and $[y']_i$. When another party receive these messages, it outputs (MULTIPLICATION-ASKED, *sid*, $vssid_1$, $vssid_2$, P_i). The parties together reconstruct y and y'. Now P_i calculates $[x]_i = [ll']_i + y[l']_i + y'[l]_i + yy'$ and outputs (MULTIPLICATION-COMPUTED, *sid*, $vssid_a$, $vssid_b$, $vssid_x$).

Using this protocol arbitrary polynomial relations can be proven analogously to the linear relations. The polynomial relations are turned into a set of multivariate polynomials which should simultaneously vanish on the committed values. To prove that a polynomial vanishes on given committed values a new commitment representing the polynomial evaluated at the given commitments is computed step by step using the above multiplication protocol as well as addition and scalar multiplication which are granted by the linearity of the VSS scheme. The new commitment is then opened to be zero.
By applying addition of shares and multiplication of shares we can obtain shares for arbitrary polynomial relations among shares. E.g. to prove that $mm' = m''$ the dealer lets the players compute $[mm'] - [m''] = [mm' - m'']$ and shows this to be zero.

For adding two shared values or for multiplication with a constant no-one has to know the shared values which are linearly transformed. Note that in DOTMP, too, no-one has to know the shared values which have to be multiplied in advance as the values y and y' are reconstructed in the protocol. Hence an arbitrary polynomial evaluation on shared values works if, and only if, $t+1$ players cooperate. This is optimal as $n-t$ players could abort the VSS scheme anyway. These linear transformations and multiplications on shared values will be the building blocks for the multiparty protocols presented in Section 5.

4 UC Security

Theorem 1. *Assuming the existence of a commodity functionality, the above protocols together UC-realizes the functionality $\mathcal{F}_{MA-VSS}$.*

We construct the ideal-protocol adversary $\mathcal{S}$ as follows. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ in a black-box way, plays the role of the ideal functionality $\mathcal{F}_{Commodity}$ and simulates a copy of the hybrid interaction of the above protocol (i.e., the real protocol with access to $\mathcal{F}_{Commodity}$) for the simulated adversary $\mathcal{A}$. In addition, $\mathcal{S}$ forwards all inputs from $\mathcal{Z}$ to $\mathcal{A}$ and all outputs from $\mathcal{A}$ to $\mathcal{Z}$.
Note that in our protocol, the access structure is composed of all sets with $t+1$ or more members. Hence, if there are more than t corrupted parties, the ideal-protocol adversary $\mathcal{S}$ knows all the secrets, and so it can perfectly emulate the execution of the real protocol for $\mathcal{A}$ (i.e., using the knowledge of the secrets, it can perfectly simulate the honest parties interacting with the real adversary $\mathcal{A}$ in the real protocol). Below we describe the procedures of the simulator when there are at most t corrupted players:

Setup: When some party (during the setup phase) demand $\mathcal{F}_{Commodity}$ to distribute the polynomials and points used in the VSS protocols and to prove polynomial relations among shared secrets, $\mathcal{S}$ simulates the actions of $\mathcal{F}_{Commodity}$ correctly and sends the outputs of the simulated $\mathcal{F}_{Commodity}$ to the parties in the simulated hybrid interaction.

Share - Honest Dealer: If some honest dealer shares a secret, the functionality $\mathcal{F}_{MA-VSS}$ sends the public delayed output (SHARED, *sid*, *vssid*) to $\mathcal{S}$. Then $\mathcal{S}$ simulates the honest dealer committing to a random secret value in the hybrid interaction. $\mathcal{S}$ lets $\mathcal{F}_{MA-VSS}$ output (SHARED, *sid*, *vssid*) to the honest parties that generated this message in the simulated hybrid interaction.

Share - Corrupted Dealer: If $\mathcal{A}$ lets some corrupted party share a secret, $\mathcal{S}$ (that plays the role of $\mathcal{F}_{Commodity}$) knows all the polynomials, so it can extract the unique value for which the secret will be reconstructed with overwhelming probability (i.e., the constant of the polynomial). $\mathcal{S}$ sends this value to $\mathcal{F}_{MA-VSS}$ and lets $\mathcal{F}_{MA-VSS}$ output (SHARED, *sid*, *vssid*) to the honest parties that generated this message in the simulated hybrid interaction.

Reconstruction - Honest Party: If some honest party wants to open a secret, $\mathcal{S}$ receives the public delayed output (OPENING-ASKED, *sid*, *vssid*, P) from $\mathcal{F}_{MA-VSS}$. Then $\mathcal{S}$ simulates the running of the reconstruction and verification procedures in the hybrid interaction. $\mathcal{S}$ chooses the share to reveal as follows:

- If the dealer is corrupted, $\mathcal{S}$ uses the share that the dealer sent in the hybrid interaction.
- If the dealer is honest and the number of parties in the opening list of this secret is less than $t+1$, $\mathcal{S}$ performs this simulation using the correct share for P (i.e., the one that it has generated playing the role of $\mathcal{F}_{Commodity}$).
- Otherwise, $\mathcal{S}$ knows all the verification functions and shares distributed to the parties as well as the polynomial distributed to the dealer in the setup phase, but $\mathcal{A}$ knows these information for at most t parties. So $\mathcal{S}$ receives the message (OPEN, *sid*, *vssid*, x_{vssid}) from $\mathcal{F}_{MA-VSS}$ and then computes the modified polynomial f' in such a way that f' is equal to f on the points that the adversary $\mathcal{A}$ knows it, but the constant of the polynomial f' is x_{vssid}. Since $\mathcal{A}$ also knows at most t points from v_i, $\mathcal{S}$ can also modify this polynomial in order to be compatible with the broadcasted value $v_i(0) + f(i)$.

If some honest party output (OPENING-ASKED, *sid*, *vssid*, P) or (OPEN, *sid*, *vssid*, x_{vssid}) in the simulated hybrid interaction, $\mathcal{S}$ allows $\mathcal{F}_{MA-VSS}$ to output the respective message to that party in the ideal protocol.

Reconstruction - Corrupted Party: If $\mathcal{A}$ lets some corrupted party P broadcast a share, $\mathcal{S}$ checks if that is the correct share for that party ($\mathcal{S}$ can do this because it plays the role of $\mathcal{F}_{Commodity}$ and so it knows all the shares distributed in the setup phase). If this condition is satisfied, $\mathcal{S}$ sends the message (OPEN, *sid*, *vssid*) to $\mathcal{F}_{MA-VSS}$ in the name of P and allows $\mathcal{F}_{MA-VSS}$ to send the message (OPENING-ASKED, *sid*, *vssid*, P) to the honest parties.

Linear Operation - Honest Party: If some honest party asks to compute a linear operation on shares, $\mathcal{S}$ receives the public delayed output (LINEAR-OPERATION-COMPUTED, *sid*, $vssid_1$, λ, $vssid_2$, *vssid*) from the functionality $\mathcal{F}_{MA-VSS}$. $\mathcal{S}$ broadcasts the message (LINEAR-OPERATION, *sid*, $vssid_1$, λ, $vssid_2$) in the simulated hybrid interaction. $\mathcal{S}$ lets $\mathcal{F}_{MA-VSS}$ deliver the message (LINEAR-OPERATION-COMPUTED, *sid*, $vssid_1$, λ, $vssid_2$, *vssid*) to the honest parties that produced this message in the simulated hybrid interaction.

Linear Operation - Corrupted Party: If $\mathcal{A}$ lets some corrupted party broadcast a message (LINEAR-OPERATION, *sid*, $vssid_1$, λ, $vssid_2$) asking to compute a linear operation on shares, $\mathcal{S}$ sends the message to $\mathcal{F}_{MA-VSS}$ and lets $\mathcal{F}_{MA-VSS}$ deliver the message (LINEAR-OPERATION-COMPUTED, *sid*, $vssid_1$, λ, $vssid_2$, *vssid*) to the honest parties in the ideal protocol.

Multiplication - Honest Party: If an honest party P tries to multiply two shares $vssid_1$ and $vssid_2$, $\mathcal{S}$ receives the public delayed output (MULTIPLICATION-ASKED, *sid*, $vssid_1$, $vssid_2$, P) from $\mathcal{F}_{MA-VSS}$. Then $\mathcal{S}$ simulates the broadcast message (MULTIPLY, *sid*, $vssid_1$, $vssid_2$) from P in the hybrid interaction and allows $\mathcal{F}_{MA-VSS}$ to send the message (MULTIPLICATION-ASKED, *sid*, $vssid_1$, $vssid_2$, P) to the honest parties if they generated this output in the simulated hybrid interaction. In addition, $\mathcal{S}$ simulates the procedures of DOTMP using the procedures described previously to deal with the intermediary secrets, and if the parties correctly reconstruct the intermediary secrets y and y', $\mathcal{S}$ lets the functionality $\mathcal{F}_{MA-VSS}$ send the message (MULTIPLICATION-COMPUTED, *sid*, $vssid_1$, $vssid_2$, *vssid*) to the honest parties that received this message in the simulated hybrid interaction.

Multiplication - Corrupted Party: If $\mathcal{A}$ lets some corrupted party P try to multiply two shares $vssid_1$ and $vssid_2$, $\mathcal{S}$ sends the message (MULTIPLY, *sid*, $vssid_1$, $vssid_2$) in the name of that party to $\mathcal{F}_{MA-VSS}$ and lets $\mathcal{F}_{MA-VSS}$ deliver the message (MULTIPLICATION-ASKED, *sid*, $vssid_1$, $vssid_2$, P) to the honest parties. In addition, $\mathcal{S}$ simulates the procedures of DOTMP using the procedures described previously to deal with the intermediary secrets, and if the parties correctly reconstruct the intermediary secrets y and y', $\mathcal{S}$ lets the functionality $\mathcal{F}_{MA-VSS}$ send the message (MULTIPLICATION-COMPUTED, *sid*, $vssid_1$, $vssid_2$, *vssid*) to the honest parties that received this message in the simulated hybrid interaction.

In order to prove $\mathcal{Z}$'s view when interacting with $\mathcal{S}$ and $\mathcal{F}$ statistically indistinguishable from its view when interacting with $\mathcal{A}$ and parties running π, we need a few auxiliary propositions.
First we analyze the operations among shared secrets (linear and multiplicative operations). The security and correctness are obvious for the linear operations among shared secrets. The following proposition proves the security and correctness for the DOTMP protocol:

Proposition 2. *For given shared values a, b it is possible to calculate shares for the value ab if and only if $t+1$ players cooperate. Especially it is possible to give zero knowledge proofs for arbitrary polynomial relations on shared values.*

We now analyze the security and correctness of our protocol for proving multiplicative relations among shares. To show that it is secure, note that the players only learn the values y and y', which give no information on a and b, since l and l' are random numbers.
To show the correctness of this protocol, we note that $[ll'] + y[l'] + y'[l] + yy' = [ll'] + (a-l)[l'] + (b-l')[l] + (a-l)(b-l') = [ll' + al' - ll' + bl - l'l + ab - lb - l'a + ll'] = [ab]$ (due to the linearity of the VSS).
We state now a proposition about the verification procedure:

Proposition 3. *With overwhelming probability the verification procedures of all honest parties will have the same result and will detect any false share.*

If the adversary change one share in the reconstruct phase, he is trying to use a polynomial $f'(x)$ in the protocol that is different from the original polynomial $f(x)$. Since by Lagrange interpolation one polynomial of degree t is completely determined by $t+1$ different points, the polynomial $f'(x)$ used by the adversary may coincide with the original polynomial in at most t points. Thus an honest player detect a false share with probability $1-\frac{t}{q}$. Therefore, with probability greater than $(1-\frac{t}{q})^n$, all honest players detect a false share. Since $\frac{t}{q}$ is negligible and n is polynomial, we have that with overwhelming probability all honest players detect a false share.

Proposition 4. *For an honest dealer and $\mathcal{A}$ that knows at most t shares of some secret, the secret (i.e., the constant of the polynomial) is completely undetermined (it can be any element in the field $GF(q)$) from the point of view of $\mathcal{A}$.*

We analyze below the probabilities of the events that can result in different views in the real execution of the protocol, with adversary $\mathcal{A}$, and in the ideal execution of the protocol, with simulator $\mathcal{S}$:

- The simulated setup procedure perfectly emulates the real protocol.
- There are at most t corrupted parties, so it follows from proposition 4 that the share procedure with honest dealer perfectly emulates the real execution of the protocol.
- From proposition 3, it follows that the procedure used in the share with corrupted dealer is statistically indistinguishable from the real protocol execution.
- From propositions 3 and 4, it follows that $\mathcal{S}$'s procedures for reconstruction with honest party are statistically indistinguishable from the real protocol execution.
- The procedure of reconstruction with corrupted party generate a view different from the real execution only if some honest party accept some false share in the simulated real protocol, but by proposition 3 this probability is negligible. So the procedure used in the reconstruction for a corrupt party is statistically indistinguishable from the real protocol execution.
- The procedures to simulate the linear operations perfectly emulates the real execution of the protocol.
- From proposition 2 and the items above, it follows that the simulator's procedures to deal with multiplication of shared secret are statistically indistinguishable from the real protocol execution.

As all events that can result in different views have negligible probabilities, the protocol UC-realizes $\mathcal{F}_{MA-VSS}$. This completes the security proof of the protocol.

5 Secure Multiparty Computations

It is known that secure multiplication and addition of shares implies secure multiparty computation [3, 17–19, 32]. Thus, our previous results can be used as building blocks for obtaining general secure computation protocols. However, for the sake of completeness, we briefly sketch, without proofs, how to put the previous results together to obtain our desired result.
Intuitively a multiparty computation is a protocol by which n interacting Turing machines can map $n-$tuples of inputs (one input held by each party) into $n-$tuples of outputs (one held by each party).
Assuming that the static adversary corrupts at most t players, our protocol statistically and UC-securely computes arbitrary polynomial relations in the commodity based model.
The secure multiparty protocols presented here have four stages. A *setup phase* where the trusted center pre-distributes data. An *input phase* where the players receive inputs and commit to these by VSS. A *computation phase* where linear transformations and multiplications are performed on shared values, but no information about the inputs is revealed. And the *opening stage* during which the results of the computation are reconstructed.

Setup Phase In this stage, the players receive from $\mathcal{F}_{Commodity}$ a circuit for securely evaluating the function on a random input. This circuit uses the verifiable secret sharing for distributing the players inputs and uses addition, scalar multiplication and multiplication gates to perform the operations. I.e., $\mathcal{F}_{Commodity}$ pre-distributes verifiable secret sharings and the multiplication gates (instances of DOTMP), since addition and scalar multiplication can be done locally by the parties that use our VSS scheme. The verification information, w_i, used in the secret sharings and the multiplication gates is the same for each party.

Input Phase The players receive inputs from $GF(q)$ and share their inputs with the given commodities for VSS as described in the main part of this paper (i.e., the players transform the pre-distributed VSS for a random value into a VSS for the desired value). As each dealer is a participant of the secure computation as well this party has to compute a share of his own from the pre-distributed data.

Computation Phase During the computation stage, the players evaluate an arithmetic circuit gate by gate using the linearity of the VSS for computing the output of the addition and scalar multiplication gates and using the DOTMP protocol for computing the output of the multiplications gates. Note that computations are necessary on intermediate results as well. Intermediate results are shared among the players, but not known to any player hence it is important that the linearity of the VSS and the DOTMP protocol can be used even if no-one knows the contents of the shared secrets involved.

Opening Phase All players reconstruct the output of the circuit using the VSS protocol and obtain the output of the function on the desired input.

References

1. B. Barak, R. Canetti, J. B. Nielsen, R. Pass. Universally Composable Protocols with Relaxed Set-Up Assumptions. *36th FOCS*, pp.186-195. 2004.
2. D. Beaver. Commodity-Based Cryptography (Extended Abstract). STOC 1997, pp. 446–455.
3. D. Beaver, S. Goldwasser. Multiparty Computation with Faulty Majority. Proc. of FOCS 1989, pp.468–473.
4. D. Beaver, S. Goldwasser. Multiparty Computation with Faulty Majority. Advances in Cryptology - CRYPTO 89, LNCS 435, 1989, pp. 589–590.
5. M. Ben-Or, S. Goldwasser, A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. 20th STOC, pp. 1–10, 1988.
6. C. Blundo, B. Masucci, D. R. Stinson, R. Wei. Constructions and Bounds for Unconditionally Secure Non-interactive Commitment Schemes. manuscript, 2001.
7. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Available at http://eprint.iacr.org/2000/067. 2005. Extended Abstract appeared in proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS), 2001.
8. R. Canetti and M. Fischlin. Universally composable commitments. *In Advances in Cryptology - Crypto 2001*, pages 19-40, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.
9. R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai. Universally Composable Two Party and Multi-party Secure Computation. *34th STOC*, pp. 494-503, 2002.
10. B. Chor, S. Goldwasser, S. Micali and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. 26th IEEE Symp. on Foundations of Computer Science, pp. 383–395, 1985.
11. R. Cramer, I. Damgard, S. Dziembowski, M. Hirt, T. Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. EUROCRYPT 99, pp. 311–326, 1999.

12. C. Crépeau, J. van de Graaf, A. Tapp. Committed Oblivious Transfer and Private multiparty Computations. CRYPTO 1995, pp. 110–123, 1995.
13. I. Damgård, J. Groth. Non interactive and reusable non-malleable commitment schemes. *34th STOC*, pp. 426-437. 2003.
14. I. Damgård and J. B. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. *CRYPTO 2002*, pp.581-596. 2002.
15. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. 28th IEEE Symp. on Foundations of Computer Science, pp. 427–437, 1987.
16. O. Goldreich, Foundations of Cryptography : Volume 1 - Basic Tools. Cambridge University Press. 2001.
17. O. Goldreich: Secure multiparty Computation, *lecture notes*, available from http://www.wisdom.weizmann.ac.il/~oded/pp.html
18. O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game or a Completeness Theorem for Protocols with Honest Majority. STOC 1987, pp. 218–229, 1987.
19. S. Goldwasser, L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. CRYPTO 1990, pp. 77–93.
20. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems, SIAM Journal on Comput., Vol. 18, No. 1, 1989, pp. 186–208.
21. G. Hanaoka, H. Imai, J. Müller-Quade, A. Nascimento, A. Otsuka. Unconditionally Secure Homomorphic Pre-distributed Bit Commitment and Secure Two-Party Computations. ISC'03.
22. G. Hanaoka, J. Shikata, Y. Zheng, H. Imai. Unconditionally Secure Digital Signature Schemes Admitting Transferability. Proc. of Asiacrypt '2000, pp. 130–142, 2000.
23. D. Hofheinz and J. Müller-Quade. Universally Composable Commitments Using Random Oracles. *Theory of Cryptography Conference (TCC)*, LNCS 2951 pp. 58-74. 2004.
24. D. Hofheinz, J. Müller-Quade, D. Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In 5th Central European Conference on Cryptology, page A version is available at http://homepages.cwi.nl/ hofheinz/card.pdf., 2005.
25. R. Dowsley, J. van de Graaf, D. Marques, A.C.A. Nascimento. A Two-Party Protocol with Trusted Initializer for Computing the Inner Product. In: Chung, Y., Yung, M. (eds.) WISA 2010. LNCS, vol. 6513, pp. 337–350. Springer, Heidelberg (2011)
26. R. Dowsley, J. Müller-Quade, A. C. A. Nascimento. On Possibility of Universally Composable Commitments Based on Noisy Channels. *SBSEG 2008*, pp. 103–114, 2008.
27. R. Dowsley, J. van de Graaf, J. Müller-Quade, A.C.A. Nascimento. On the Composability of Statistically Secure Bit Commitments. Cryptology ePrint Archive, Report 2008/457 (2008), http://eprint.iacr.org/.
28. J. Katz. Universally Composable Multi-party Computation Using Tamper-Proof Hardware. *EUROCRYPT 2007*. pp. 115–128. 2007.
29. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. ACM Trans. on Programming Languages and Systems, 4(3), pp. 382– 401, July 1982.
30. T. Matsumoto and H. Imai. On the Key Pre-distribution Systems: A Practical Solution to the Key Distribution Problem. CRYPTO 1987, pp. 185–193, 1988.
31. R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. Communications of the ACM, 24, pp. 583–584, 1981.
32. S. Micali and P. Rogaway. Secure Computation (Abstract). CRYPTO 1991: pp. 392–404, 1991.
33. T. Moran and G. Segev. David and Goliath Commitments: UC Computation for Asymmetric Parties Using Tamper-Proof Hardware. *EUROCRYPT 2008*. pp. 527–544. 2008.
34. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. CRYPTO '91, pp. 129–140. Springer-Verlag, 1991.
35. M. Prabhakaran and A. Sahai. New Notions of Security: Achieving Universal Composability without Trusted Setup. In *Proc. of STOC*, 2004.
36. T. Rabin, M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. Proc. ACM STOC '89, pp. 73–85, ACM Press, 1989.

37. R.L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using concealing channels and a trusted initializer. Manuscript, 1999.
38. M. Tompa and H. Wolf. How to share a secret with cheaters. Journal of Cryptology, 1(2):133-138, 1988.