Non-Interactive Verifiable Secret Sharing For Monotone Circuits

Ge Bai[1], Ivan Damgård[2], Claudio Orlandi[2], and Yu Xia[1]

[1] IIIS, Tsinghua University[⋆]
[2] Aarhus University

Abstract. We propose a computationally secure and *non-interactive* verifiable secret sharing scheme that can be *efficiently* constructed from any monotone Boolean circuit. By non-interactive we mean that the dealer needs to be active only once, where he posts a public message as well as a private message to each shareholder. In the random oracle model, we can even avoid interaction between shareholders. By efficient, we mean that we avoid generic zero-knowledge techniques. Such efficient constructions were previously only known from linear secret sharing schemes (LSSS). It is believed that the class of access structures that can be handled with polynomial size LSSS is incomparable to the class that can be recognized by polynomial size monotone circuits, so in this sense we extend the class of access structures with efficient and non-interactive VSS.

1 Introduction

Secret-Sharing. Secret sharing schemes are fundamentally important tools in many areas of cryptography, because they allow us to strike a balance between confidentiality and security against loss of data, by storing shares of the data in separate locations. This is useful, e.g., when storing cryptographic keys.

A secret sharing scheme is defined by two algorithms: a probabilistic *sharing* algorithm which takes a secret message m as input and produces n shares $s_1, ..., s_n$; and a *reconstruction* algorithm that takes a subset of shares $\{s_i | i \in I\}$ as input and outputs m, provided I is a *qualified* set. The family of qualified sets is called the *access* structure of the scheme. An access structure $\mathcal{A}$ is always monotone: if $I \in \mathcal{A}$ and $I \subset J$ then $J \in \mathcal{A}$. We also require that if a set J is not qualified, then the subset of shares $\{s_i | i \in J\}$ gives no information on m. More precisely, in a perfect scheme, such a subset has distribution independent of m. Or in case of *computational* secret sharing, an unqualified set of shares has a distribution that can be simulated with computationally indistinguishable distribution without knowing m. In a perfect scheme, shares must be at least as large as the secret, while in computational secret sharing they can be much smaller, which is a main motivation for considering computational schemes.

Naturally, we want the total size of the shares to be minimal, and certainly polynomial in n, and also we would like the sharing algorithm to run in time polynomial in n.[3]

Access Structures. In the following, we will speak of the access structure *characterised* by (for instance) a monotone Boolean circuit with n input bits: we think of the input bits as being in 1-1 correspondence with n players, and a player subset I can then be translated to a bit string by setting the bits corresponding to members of I to 1 and the rest to 0. The access structure characterised by the formula now consists of those subsets whose corresponding bit string causes the formula to output 1. This notion generalises naturally to any other computational model that computes monotone functions of bit strings.

Verifiable Secret-Sharing. One well-known and natural extension of secret sharing is *verifiable* secret sharing (VSS), where the party generating the shares (called the *dealer* in the following), and in addition some unqualified subset of the players (or shareholders) may be corrupted by a malicious adversary. We now execute a protocol in which the dealer sends shares to the players and some verification is performed. If the

[⋆] Work done while visiting Aarhus University.
[3] However, since there are doubly exponentially many (families of) access structures, an easy counting argument shows that we cannot hope to handle all access structures with polynomial time sharing algorithms.

sharing phase is successful, all honest players must output a share, and otherwise they all reject. Later, any qualified player subset can go together and reconstruct the secret.

We now want the following properties: if the dealer is corrupt and the sharing phase is successful, the secret is well defined in the sense that any qualified subset will later reconstruct the same secret. If the dealer is honest, the sharing is always successful, and the secret is what the dealer intended; furthermore no unqualified player subset has any information on the secret (information theoretically or computationally). A VSS can be seen as a distributed commitment scheme that allows to open the committed information even if the dealer is not present.

This is an important property and can prevent different attacks depending on the scenario: one can imagine using a VSS as a distributed commitment scheme where we want the commitment to be binding. Think of a powerful cheating dealer who can "shut down" players arbitrarily; VSS guarantees that such a dealer cannot change its mind and control the output of the reconstruction by shutting down selected players. A perhaps more practically oriented application is in multiparty computation: consider a client with some secret input that he wants to supply to a multiparty computation run by a set of servers. A natural solution that will handle malicious attacks is to VSS the input among the servers. However, if this takes place in an asynchronous environment like the Internet, we clearly need a scheme with as little interaction as possible, ideally a client should be able to just post some information and then leave.

Non-Interactive VSS. The amount of interaction needed for the sharing phase of a VSS depends on the model for communication that is assumed. In this paper, we focus on the model where players have public encryption keys and hold corresponding secret keys, and where no secure channels are assumed to be given "for free". But we assume that the dealer can publish information that all honest players will agree on (using, e.g., a bulletin-board or a broadcast channel). In this model, we clearly cannot get security without making computational assumptions. Moreover, the best we can hope for in terms of interaction is that the dealer publishes a single message, and each player then computes his output from this message and his secret key. We call such a scheme *non-interactive.* A slightly stronger property that many *non-interactive* schemes are born to satisfy is *public verifiability.* Namely, anyone, even outside the scheme, but with access to the public information, can perform the verification. One may also aim for a weaker property called *non-interactive with complaints*, where the scheme is only non-interactive if corrupt players behave honestly – but (motivated by the Internet scenario explained above) we require that even if interaction is needed, the dealer does not need to take part in this.

Our Contributions. In this paper, we present two VSS protocols with computational security, the first is non-interactive with complaints, while the second is non-interactive and *publicly verifiable* in the random oracle model (used for the Fiat-Shamir heuristic). Both schemes are built on top of the same *locally verifiable* scheme, which is our main technical contribution and is based on the standard RSA assumption.

The complexity of the scheme is polynomial in the size of a monotone Boolean circuit characterising the access structure. It is the first scheme with this property and it allows us to efficiently handle access structures that cannot be done efficiently with linear schemes. We emphasise that although we assume that some set-up information is available, we do not use generic non-interactive zero-knowledge (which could be used to solve the problem in a rather trivial way). In particular the communication complexity of our scheme does not depend on the circuit complexity of the dealer's computation when he generates shares, only on the security parameter and the number of fan-outs in the circuit characterising the access structure.

Related Work. The notion of information theoretic secret sharing was independently discovered by Blakley [Bla79] and by Shamir [Sha79], who constructed efficient schemes for simple threshold access structures. The introduction of general secret sharing is due to Ito, Saito and Nishiziki [ISN89]. Later, Benaloh and Leichter [BL90] constructed schemes that are efficient in the size of a Boolean formula characterising the target access structure. Karchmer and Wigderson [KW93] introduced the notion of a monotone linear span program (MSP) and showed that any MSP induces a linear secret sharing scheme (with complexity polynomial in the size of the MSP) for the access structure characterised by the MSP. In fact, any linear secret sharing scheme can be seen as being derived from an MSP, including the schemes by Shamir and Benaloh-Leichter.

This gives us efficient secret sharing schemes for any access structure that can be characterized by an MSP of size polynomial in the number of players. If one now wants to extend the class of access structures we can handle efficiently, it is natural to consider those characterized by polyomial size Boolean *circuits* (rather than formulas as considered in [BL90]). The reason for this is that the classes of access structures characterised by polynomial size MSPs, respectively monotone Boolean circuits are incomparable as far as we know, and hence being able to construct secret sharing schemes efficiently from monotone circuits would indeed give us something new. However, we know no such construction resulting in a perfect scheme. On the other hand, Vinod et al. [VNS$^+$03] proposed a construction yielding computational security (based on an unpublished idea by Yao [Yao89]).

As for construction of verifiable secret sharing schemes, all constructions we know of are schemes that start from a regular secret sharing scheme and add verifiability on top. In a model where there are secure point-to-point channels between all pairs of players, this can be done with perfect security for any linear scheme, under certain conditions on the access structure [CDM00]. One can even convert any secret sharing in a black-box fashion to VSS with statistical security [CDD00] but the construction uses a lot of interaction and generic zero-knowledge techniques to some extent. In the communication model we consider in this paper, any linear scheme cane made verifiable under computational assumptions. The basic idea was first proposed by Pedersen [Ped92] based on Shamir's scheme, but the principle easily extends to any linear scheme. The resulting VSS is "almost" non-interactive with complaints, i.e., the dealer needs to help resolve conflicts if there are complaints. This problem was resolved in [DT07]. The notion of publicly verifiable secret sharing is introduced by Stadler in [Sta96].

Technical Overview. Our main result is to extend the class of access structures we can handle efficiently and verifiably, in the same way that [VNS$^+$03] extended what we can do with regular secret sharing. We do so by constructing a verifiable version of the scheme from [VNS$^+$03][4]. Prior to our work, no such construction was known.

The idea is that the dealer runs (a version of) the sharing scheme from [VNS$^+$03], resulting in each player receiving his share. But in addition he also makes public a "tag" for each share (which is constructed in such a way as to preserve computational privacy). Every shareholder can check the share they received against the public tag. Furthermore, it is now possible (very simplistically speaking) to run the reconstruction algorithm on the set of all tags and make checks under way such that if the "homomorphic reconstruction" does not abort, then the set of actual shares is indeed a consistent sharing of a well defined secret.

We obtain this by constructing a symmetric encryption scheme (as required in the secret sharing scheme from [VNS$^+$03]) and a tagging scheme that "lives" in the same RSA group and has certain convenient homomorphic properties.

This does not yet ensure that everything will always be fine: the dealer could make a good set of tags, but send an incorrect share to one or more of the players. A player P can detect that he has been sent a bad share, but this will not be clear to the other players: they have only seen a ciphertext meant for P and cannot tell if the dealer is corrupt or P is complaining for no good reason.

We can resolve this in two different ways, in both cases by using a specific encryption scheme for sending shares to players. The first approach is to use an encryption scheme that has so-called verifiable opening, allowing P to reveal his share along with a proof that this was indeed what the dealer sent him in encrypted form. This technique was introduced in [DT07] and formalized in [DHKT08]. This gives a scheme that is non-interactive with complaints. The second approach is to use an encryption scheme we suggest that is designed to allow the dealer to give an efficient proof (in the random oracle model) that the shares he encrypts indeed correspond to the tags he publishes. This gives a non-interactive scheme with only a small computational overhead compared to the work needed to compute shares in a non-verified way.

Roadmap. Section 2 introduces the notation and some preliminaries which will be used in the rest of the paper. Section 3 briefly reviews the computational secret sharing scheme of Vinod et al. Section 4 introduces

[4] On the way to our result, as a secondary contribution, we also prove that the construction of [VNS$^+$03] satisfies a strong, simulation based notion of privacy, while the original paper only argues for a weaker, "one-way" definition of privacy.

the notion of locally verifiable secret sharing and presents our novel construction. Finally in Section 5 we discuss how to combine the locally verifiable scheme with different kind of encryption schemes to achieve a *non-interactive publicly verifiable* scheme and a *non-interactive scheme with complaints*.

2 Notation and Preliminaries

Basic Notation. We use the shorthand $[m, n]$ for $\{m, m+1, \ldots, n-1, n\}$ and $[n]$ for $[1, n]$. If S is a set $x \leftarrow S$ is a random element from S, if A is an algorithm $y \leftarrow A(x)$ is the output of A run on input x on an uniformly random tape. If $S \subseteq [n]$, then $s = \mathsf{set2bits}(S)$ is an n-bit string where the i-th bit of s is 1 iff $i \in S$. We call a function $f : \mathbb{N} \to \mathbb{R}$ *negligible* if for all c, for all big enough κ, $f(\kappa) < \kappa^{-c}$. We use $\mathsf{negl}(\kappa)$ for a generic negligible function. We say two families of distributions $U_0 = \{U_{0,\kappa}\}_{\kappa \in \mathbb{N}}, U_1 = \{U_{1,\kappa}\}_{\kappa \in \mathbb{N}}$ are *computationally indistinguishable* if for all probabilistic polynomial time (PPT) distinguisher D, $|\Pr[D(U_{b,\kappa}) = b] - \frac{1}{2}| = \mathsf{negl}(\kappa)$ and we write $U_0 \approx_c U_1$ for short.

Access Structure. An access structure $\mathcal{A}$ of $[n]$ is a monotone subset $\mathcal{A} \subseteq 2^{[n]}$. Given a set $I \subseteq [n]$ we say that I is a *qualified set* if $I \in \mathcal{A}$ or that I is an *unqualified set* if $I \notin \mathcal{A}$. We say that a Boolean circuit $C : \{0,1\}^n \to \{0,1\}$ *describes* $\mathcal{A}$ if $C(\mathsf{set2bits}(I)) = 1 \Leftrightarrow I \in A$.

Circuits. We use the following notation for circuits: a *circuit* $C : \{0,1\}^n \to \{0,1\}$ is described by $\ell > n$ wires $\{w_1, \ldots, w_\ell\}_{i \in [\ell]}$ and λ gates $\{g_i\}_{i \in [\lambda]}$. We call $\{w_i\}_{[1,n]}$ the *input wires*, $\{w_i\}_{[n+1,\ell-1]}$ the *internal wires* and we call w_ℓ the *output wire*. Every internal wire connects exactly two gates, while the input wires and the output wire are only connected to one gate (therefore, there are exactly $\lambda = (2\ell + n + 1)/3$ gates). Wires carry values: at the beginning all wires are initialized to $\perp$, and we say a wire is *assigned* if $w_i \neq \perp$.

Each *gate* g_i is described by a tuple from $[\ell]^3$ (representing pointers to their input/output wires) and a type $\mathsf{type}_i \in \{\mathsf{and}, \mathsf{or}, \mathsf{fanout}\}$, which determines the semantic of the wires: if g_i has type $\mathsf{type}_i \in \{\mathsf{and}, \mathsf{or}\}$, then g_i has two input wires $(w_{i_{LI}}, w_{i_{RI}})$ (for left input and right input) and one output wire w_{i_O}. Finally, if g_i has type $\mathsf{type}_i = \mathsf{fanout}$, then g_i has one input wire w_{i_I} and two output wires $(w_{i_{LO}}, w_{i_{RO}})$. We assume (without loss of generality) that the output wire w_ℓ is not the output of a fan-out gate.

We say that a gate is *ready to be evaluated* if: both $w_{i_{LI}} \neq \perp$ and $w_{i_{RI}} \neq \perp$ when $\mathsf{type}_i = \mathsf{and}$; either $w_{i_{LI}} \neq \perp$ or $w_{i_{RI}} \neq \perp$ when $\mathsf{type}_i = \mathsf{or}$; $w_{i_I} \neq \perp$ when $\mathsf{type}_i = \mathsf{fanout}$. Finally, we say that a gate is *assigned* if all its output wires have been assigned.

The type of a gate determines how a circuit is evaluated. To exemplify our notation, we describe how to evaluate a simple Boolean circuit $C : \{0,1\}^n \to \{0,1\}$ on input $x \in \{0,1\}^n$.

1. Parse the input $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ and assign $w_i = x_i$ for all $i \in [n]$;
2. While $w_\ell = \perp$, find the first gate g_i which is *ready to be evaluated*:
   – If $\mathsf{type}_i = \mathsf{and}$: Assign $w_{i_O} = w_{i_{LI}} \wedge w_{i_{RI}}$;
   – If $\mathsf{type}_i = \mathsf{or}$: Assign $w_{i_O} = w_{i_{LI}} \vee w_{i_{RI}}$;
   – If $\mathsf{type}_i = \mathsf{fanout}$: Assign $w_{i_{LO}} = w_I$ and $w_{i_{RO}} = w_I$;
3. Output w_ℓ;

Finally, we call $(\ell_{\mathsf{and}}, \ell_{\mathsf{or}}, \ell_{\mathsf{fanout}})$ respectively the number of $(\mathsf{and}, \mathsf{or}, \mathsf{fanout})$ gates in the circuit.

3 Computational Secret Sharing

In this section we review the basic definitions of a computational secret-sharing scheme and present the construction of Vinod et al. [VNS+03] in our notation. A *computational secret-sharing scheme* (CS3) is a tuple of algorithms $\pi = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Rec})$ which are defined and used as follows:

Setup: The randomized setup algorithm $pp \leftarrow \mathsf{Setup}(1^\kappa)$ (run once and for all) outputs some public system parameters pp for the secret sharing scheme (which contains, among other things, some message space $\mathcal{M}$ from which the secret can be chosen)[5].

[5] In case where no trusted party exists to run this setup, a secure computation protocol can be used instead. We note that our setup algorithm will output an RSA modulus, and that several efficient protocols for this task exist, depending on the desired security guarantees and threshold.

Share: A dealer can share a secret message $m \in \mathcal{M}$ with n parties $P_1, \ldots, P_n$ according to an access structure described by a circuit $C : \{0,1\}^n \to \{0,1\}$ by running the randomized algorithm $(s_0, s_1, \ldots, s_n) \leftarrow \mathsf{Share}_{pp}(C, m)$ which outputs $n+1$ shares $s_0, s_1, \ldots, s_n$. The dealer sends to each party P_i the shares (s_0, s_i). Sometimes we refer to s_0 as the *public share* and to the s_i's, $i \in [n]$ as the private shares.

Reconstruct: A set of parties $\{P_i\}_{i \in I}$ such that $C(\mathsf{set2bits}(I)) = 1$ can reconstruct the secret message m by running $m \leftarrow \mathsf{Rec}_{pp}(s_0, \{s_i\}_{i \in I})$.

Intuitively, we want such a scheme to be *correct* (any qualified set of parties can reconstruct the secret m) and *private* (any unqualified set of parties does not learn any information about m). This can be formalized as follows:

Definition 1 (Correctness). *A CS3 π is* correct *if for all $m \in \mathcal{M}$, for all circuits C describing an access structure $\mathcal{A}$, and for all $I \in \mathcal{A}$,*

$$\Pr[m \neq \mathsf{Rec}_{pp}(s_0, \{s_i\}_{i \in I})] \leq \mathsf{negl}(\kappa)$$

Where $(s_0, s_1, \ldots, s_n) \leftarrow \mathsf{Share}_{pp}(C, m), pp \leftarrow \mathsf{Setup}(1^\kappa)$ and the probabilities are taken over the choices of all algorithms.

In the following definition we ask for privacy in a strong, simulation based sense, while in the original work [VNS+03] only a weaker "one-way" version of privacy is considered.

Definition 2 (Privacy). *A CS3 π is* private *if for all circuits C describing an access structure $\mathcal{A}$, and for all $I \notin \mathcal{A}$, there exist a PPT simulator* Sim *such that for all $m \in \mathcal{M}$:*

$$\mathsf{Sim}(pp, C, I) \approx_c (s_0, \{s_i\}_{i \in I})$$

where $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(s_0, s_1, \ldots, s_n) \leftarrow \mathsf{Share}_{pp}(C, m)$

Remarks on the model. Note that at this point we make the assumption that there are secure point-to-point channels between the dealer and the parties. This assumption will be removed in Section 5 where we will show two techniques for distributing the shares which make the overall scheme verifiable against a malicious dealer.

Constructing CS3s. Vinod et al. [VNS+03] proposed a CS3 with the following communication complexity: $|s_0| = O(|C| + \kappa \cdot \ell_{\mathsf{fanout}})$ and $s_i = O(\kappa)$ for all $i \in [n]$. Their scheme uses a symmetric encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$, where the key space and message space of the encryption scheme are the group used by the secret sharing scheme (for instance, the group of κ-bit strings with bitwise XOR of strings as the group operation).

Setup: The setup algorithm outputs a cyclic group $\mathbb{G}$ (which we write here in multiplicative notation) which is used both as the message space $\mathcal{M}$ and as the working group for the scheme, as well as an IND-CPA secure symmetric encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ where the key space and the message space is $\mathbb{G}$.

Share: To share a secret $m \in \mathbb{G}$ with an access structure described by a circuit C among n parties, the dealer runs the following algorithm:

1. Assign $w_\ell = m$ and let $s_0 = C$;
2. While $w_j = \perp$ for some $j \in [n]$ (the input wires), find the first *assigned* gate g_i and:
   - If $\mathsf{type}_i = \mathsf{and}$: Secret share the value of the output wire between the two input wires i.e., pick a random $w_{i_{LI}} \leftarrow \mathbb{G}$ and assign $w_{i_{RI}} = w_{i_O} \cdot (w_{i_{LI}})^{-1}$;
   - If $\mathsf{type}_i = \mathsf{or}$: Copy the value of the output wire to both input wires i.e, assign $w_{i_{LI}} = w_{i_O}$ and $w_{i_{RI}} = w_{i_O}$;
   - If $\mathsf{type}_i = \mathsf{fanout}$: Assign a fresh key to the input wire, and append encryptions of the output wires to the public share i.e., compute $k \leftarrow \mathsf{G}(1^\kappa)$, $c_{i_{LO}} = \mathsf{E}(k, w_{i_{LO}})$ and $c_{i_{RO}} = \mathsf{E}(k, w_{i_{RO}})$; finally, assign $w_{i_I} = k$ and let $s_0 = s_0||(i, c_{i_{LO}}, c_{i_{RO}})$;
3. Let $s_i = w_i$ for all $i \in [n]$, and output $(s_0, s_1, \ldots, s_n)$;

Reconstruct: To reconstruct a secret m given a set of shares $(s_0, \{s_i\}_{i\in I})$ from some qualified set I run the following algorithm:

1. Assign $w_i = s_i$ for all $i \in I$ and recover C from s_0.
2. While $w_\ell = \bot$, find the first gate g_i which is *ready to be evaluated* and:
   - If $\mathsf{type}_i = \mathsf{and}$: Assign $w_{i_O} = w_{i_{LI}} \cdot w_{i_{RI}}$;
   - If $\mathsf{type}_i = \mathsf{or}$: Assign $w_{i_O} = w_{i_{LI}}$ if $w_{i_{LI}} \neq \bot$ or $w_{i_O} = w_{i_{RI}}$ otherwise;
   - If $\mathsf{type}_i = \mathsf{fanout}$: Recover $(i, c_{i_{LO}}, c_{i_{RO}})$ from s_0 and assign

$$w_{i_{LO}} \leftarrow \mathsf{D}(w_{i_I}, c_{i_{LO}}) \text{ and } w_{i_{RO}} \leftarrow \mathsf{D}(w_{i_I}, c_{i_{RO}})$$

3. Output w_ℓ;

The scheme is correct by inspection. Privacy can be proven by constructing a simulator Sim who runs the sharing scheme for a random secret m', and then arguing that any distinguisher can be used to break the IND-CPA security of the underlying encryption scheme. We will prove this in detail for the locally verifiable variant of this scheme construction (see proof of Theorem 1).

4 Locally Verifiable Secret-Sharing Scheme

In this section we show how to make the CS3 of Vinod et al. [VNS+03] *verifiable* i.e., even if the dealer is corrupt, we want to make sure that the secret message is well defined. In particular, we need to make sure that the output of the reconstruction phase does not depend on which of the possibly many set of qualified parties reconstructs the secret. Thus we define a *locally verifiable* computational secret-sharing scheme (VCS3) by adding an algorithm as follows:

Verification: The randomized algorithm $f \leftarrow \mathsf{Ver}_{pp}(s_0, s_i)$ outputs a flag bit $f \in \{\mathtt{true}, \mathtt{false}\}$ which indicates whether (s_0, s_i) is a valid share for party P_i.

We now ask the following property:

Definition 3 (Local Verifiability). *We say a CS3 scheme is* locally verifiable *if for all $n \in \mathbb{Z}$, for all circuits C describing an access structure, and for all PPT algorithms D^* the following holds: Let $pp \leftarrow \mathsf{Setup}(1^\kappa)$ and $(s_0, s_1, \ldots, s_n) \leftarrow D^*(pp)$. If $\forall i \in [n], \mathsf{Ver}_{pp}(s_0, s_i) = \mathtt{true}$, then with a overwhelming probability there exists a value $m \in \mathcal{M}$ such that $\mathsf{Rec}_{pp}(s_0, \{s_i\}_{i\in I}) = m$ for all qualified sets $I \in \mathcal{A}$ (where $\mathcal{A}$ is defined in s_0).*

Remarks on the Model. It is clear that without some degree of interaction between the parties, it is impossible to achieve even a locally verifiable scheme. Think of a setting with two parties P_1, P_2 and a simple access structure $\mathcal{A} = \{\{P_1\}, \{P_2\}\}$. Now the dealer can simply send $s_1 \neq s_2$ to the two parties, which will therefore reconstruct to two different secrets. Therefore in this section we make the assumption that there is a broadcast channel from the dealer to the parties, which ensures that all parties P_i receive the same public share s_0. The private shares s_i's are still sent over private channels from the dealer to the parties. Note that local verifiability does not say anything about what to do when one of the parties rejects her share. We deal with complaints later in Section 5.

Feasibility of locally verifiable CS3. Note that it would be possible to enhance any CS3 scheme (such as the one presented in Section 3) with the local verifiability property described above by sending to each party, together with s_i, a non-interactive zero-knowledge proof (NIZK) that s_i was computed correctly (this could be achieved by generating the crs for the NIZK during the setup phase, and letting the dealer append a commitment to m and the randomness used in Share, and then the Ver algorithm simply checks the NIZK). However the communication of the resulting verifiable scheme would depend on the dealer's local computation (i.e., the circuit complexity of the original Share algorithm) and thus add a very significant overhead. In the following, we look for a solution which avoids this problem and essentially preserves both the communication and computational complexity of the original scheme by Vinod et al.

Locally Verifiable CS3. We present here our locally verifiable CS3. The scheme is based on the standard RSA assumption. Intuitively, we make the scheme locally verifiable by having the dealer publish some "tags" of all the private shares in the public share, using some function $t_i = \mathsf{Tag}(s_i)$. Now, since Tag is deterministic, every party can check that their private share is consistent with the public tag. In addition, the Tag function and the Rec_{pp} function are designed so that they "commute", meaning that (from a very high level point of view) it is possible to compute the reconstruction function on the tags (instead of the actual values) and verify if the obtained tag is equal to the published one i.e.,

$$\mathsf{Tag}(m) = \mathsf{Rec}_{pp}(s_0, \mathsf{Tag}(s_1), \ldots, \mathsf{Tag}(s_n))$$

4.1 Building Blocks

The group. The scheme works in an RSA group $\mathbb{Z}_N^*$ where the RSA modulus is generated during the setup phase (hence its factorization is unknown to both the dealer and the parties).[5] All operations are carried out in the group $\mathbb{Z}_N^*$, hence if $x \in \mathbb{Z}_N^*$ and $e \in \mathbb{Z}$ we write "$y = x^e$" instead of "$y = x^e \bmod N$".

The tags. The scheme uses a "tag" function $\mathsf{Tag}(x) = x^\tau$ where τ is a prime number larger than N which is generated by the dealer – it is easy to see that Tag is multiplicatively homomorphic i.e., $\mathsf{Tag}(x) \cdot \mathsf{Tag}(y) = \mathsf{Tag}(x \cdot y)$.

The encryption scheme. We also use a symmetric encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ where G outputs a random $k \leftarrow \mathbb{Z}_N^*$; The encryption function $c \leftarrow \mathsf{E}(k, m)$ chooses a random prime $\rho > n$ and outputs it together with $\sigma = k^\rho \cdot m$; The decryption function $m \leftarrow \mathsf{D}(k, c)$ outputs $m = \sigma \cdot k^{-\rho}$. We note a useful property of our encryption scheme and the Tag function, namely that they commute nicely: if $\mathsf{D}(k, (\rho, \sigma)) = m$ then $\mathsf{D}(\mathsf{Tag}(k), \rho, \mathsf{Tag}(\sigma)) = \mathsf{Tag}(m)$ since

$$\begin{aligned} \mathsf{D}(\mathsf{Tag}(k), \rho, \mathsf{Tag}(\sigma)) &= \mathsf{Tag}(\sigma) \cdot \mathsf{Tag}(k)^{-\rho} = \sigma^\tau \cdot (k^\tau)^{-\rho} = (\sigma \cdot k^{-\rho})^\tau \\ &= \mathsf{Tag}(m) \end{aligned} \tag{1}$$

Later in the proof we will need the following property from this scheme (intuitively, the Lemma says that the scheme is *one-way secure under single-query chosen plaintext attack*):

Lemma 1. *Consider the following game: 1) a challenger runs $k \leftarrow \mathsf{G}(1^\kappa)$ 2) the adversary picks a value $m \in \mathbb{Z}_N^*$; 3) the challenger picks a random $r \in \mathbb{Z}_N^*$ and sends $(\mathsf{E}(k, m), \mathsf{E}(k, r))$ to the adversary; 4) the adversary outputs r'; For all PPT adversary, $r' \neq r$ except with negligible probability if the RSA problem is hard.*

Proof. In step 3) the adversary receives a 4-tuple from $\mathbb{Z}_N^*$ composed of

$$(\rho_0, \sigma_0 = k^{\rho_0} \cdot m, \rho_1, \sigma_1 = k^{\rho_1} \cdot r)$$

We first claim that an adversary who computes $r' = r$ can be used to compute k efficiently in the following way: let a, b be the values such that $a \cdot \rho_0 + b \cdot \rho_1 = 1$ (which are guaranteed to exist since ρ_0, ρ_1 are different primes). Then

$$(\sigma_0 \cdot m^{-1})^a \cdot (\sigma_1 \cdot r^{-1})^b = k^{a \cdot \rho_0 + b \cdot \rho_1} = k$$

Now the reduction solves the RSA instance $(e, y = x^e)$ by setting $\rho_0 = e, \sigma_0 = m \cdot y$, sampling a random prime ρ_1 and a random element σ_1 from $\mathbb{Z}_N^*$. Note that this is the exact distribution that the adversary was expecting since this is equivalent to the choice of a random $r = y^{-1} \cdot \sigma_1$ in the game, and r is uniformly distributed in $\mathbb{Z}_N^*$ (unless some of the random choices are not invertible mod N, but in that case the reduction can trivially factor N).[6]

[6] (Note that the scheme would not be secure if the adversary could make 2 CPA queries, since in that case it could recover k in the same way as the reduction does.)

The extractor. Finally, $\mathsf{Ext} : \mathbb{Z}_N^* \to \{0,1\}^\mu$ is an extractor which extracts the $\mu = \log(\kappa)$ hard-core bits of the RSA function from some value in $\mathbb{Z}_N^*$ (the least significant $\log(\kappa)$ bits will do [ACGS88]).

4.2 The Construction

We are now ready to give the details of our construction:

Setup: Generate an RSA modulus N and a random prime number $\tau > N$ which defines the function $\mathsf{Tag}(x) = x^\tau$ and output $pp = (N, \tau)$;

Share: To share a secret $m \in \{0,1\}^\mu$ with an access structure described by a circuit C among n parties, the dealer runs the following algorithm.

1. Assign $w_\ell \leftarrow \mathbb{Z}_N^*$
2. Compute $u = \mathsf{Ext}(w_\ell) \oplus m$;
3. Compute $t_\ell = \mathsf{Tag}(w_\ell)$;
4. Let $s_0 = (C, u, \tau, t_\ell)$;
5. While $w_j = \perp$ for some $j \in [n]$ (the input wires), find the first *assigned* gate g_i and:
   - If $\mathsf{type}_i = \mathsf{and}$: Share the value of the output wire between the two input wires i.e., pick a random $w_{i_{LI}} \leftarrow \mathbb{Z}_N^*$ and assign $w_{i_{RI}} = w_{i_O} \cdot (w_{i_{LI}})^{-1}$; In addition, compute the tags for two input wires $t_{i_{LI}} = \mathsf{Tag}(w_{i_{LI}})$ and $t_{i_{RI}} = \mathsf{Tag}(w_{i_{RI}})$;
   - If $\mathsf{type}_i = \mathsf{or}$: Copy the value of the output wire to both input wires i.e., assign $w_{i_{LI}} = w_{i_O}$ and $w_{i_{RI}} = w_{i_O}$; In addition, copy $t_{i_{LI}} = t_{i_O}$ and $t_{i_{RI}} = t_{i_O}$;
   - If $\mathsf{type}_i = \mathsf{fanout}$: Assign a fresh key to the input wire, and append encryptions of the output wires to the public share i.e., compute $k \leftarrow \mathsf{G}(1^\kappa)$, $c_{i_{LO}} \leftarrow \mathsf{E}(k, w_{i_{LO}})$ and $c_{i_{RO}} \leftarrow \mathsf{E}(k, w_{i_{RO}})$; finally, assign $w_{i_I} = k$, compute $t_{i_I} = \mathsf{Tag}(w_{i_I})$ and let $s_0 = s_0||(c_{i_{LO}}, c_{i_{RO}}, t_{i_{LO}}, t_{i_{RO}})$;
6. Let $s_i = w_i$ for all $i \in [n]$, append $s_0 = s_0||(t_1, \ldots, t_n)$;

Verification: Upon receiving (s_0, s_i) party P_i runs the algorithm $\mathsf{Ver}_{pp}(s_0, s_i)$ described here:

1. From s_0, recover (C, u, τ, t_ℓ), t_i for every $i \in [n]$, and $c_{i_{LO}}, c_{i_{RO}}, t_{i_{LO}}, t_{i_{RO}}$ for every fanout gate g_i;
2. If $t_i \neq \mathsf{Tag}(s_i)$ stop and output `false`; else:
3. Assign $w_i = t_i$ for all $i \in [n]$.
4. While $w_\ell = \perp$, find the first gate g_i whose input wires are all assigned and:
   - If $\mathsf{type}_i = \mathsf{and}$: Assign $w_{i_O} = w_{i_{LI}} \cdot w_{i_{RI}}$;
   - If $\mathsf{type}_i = \mathsf{or}$: If $w_{i_{LI}} \neq w_{i_{RI}}$ stop and output `false`; else assign $w_{i_O} = w_{i_{LI}}$;
   - If $\mathsf{type}_i = \mathsf{fanout}$: Parse $(c_{i_{LO}}, c_{i_{RO}}) = (\rho_{i_{LO}}, \sigma_{i_{LO}}, \rho_{i_{RO}}, \sigma_{i_{RO}})$. If $\mathsf{D}(w_{i_I}, (\rho_{i_{LO}}, \mathsf{Tag}(\sigma_{i_{LO}})) \neq t_{i_{LO}}$ or $\mathsf{D}(w_{i_I}, (\rho_{i_{RO}}, \mathsf{Tag}(\sigma_{i_{RO}})) \neq t_{i_{RO}}$ stop and output `false`; else assign $w_{i_{LO}} = t_{i_{LO}}$ and $w_{i_{RO}} = t_{i_{RO}}$.
5. Stop and output `false` if $w_\ell \neq t_\ell$;
6. Else output `true`;

Reconstruct: To reconstruct a secret m given a set of shares $(s_0, \{s_i\}_{i \in I})$ from some qualified set I run the following algorithm.

1. Assign $w_i = s_i$ for all $i \in I$ and recover C from s_0.
2. While $w_\ell = \perp$, find the first gate g_i which is *ready to be evaluated* and:
   - If $\mathsf{type}_i = \mathsf{and}$: Assign $w_{i_O} = w_{i_{LI}} \cdot w_{i_{RI}}$;
   - If $\mathsf{type}_i = \mathsf{or}$: If $w_{i_{LI}} \neq \perp$, assign $w_{i_O} = w_{i_{LI}}$, else $w_{i_O} = w_{i_{RI}}$.
   - If $\mathsf{type}_i = \mathsf{fanout}$: Recover $(c_{i_{LO}}, c_{i_{RO}})$ from s_0 and assign

$$(w_{i_{LO}}, w_{i_{RO}}) \leftarrow (\mathsf{D}(w_{i_O}, c_{i_{LO}}), \mathsf{D}(w_{i_I}, c_{i_{RO}}))$$

3. Output $m = u \oplus \mathsf{Ext}(w_\ell)$;

The scheme satisfies *correctness* by inspection. We here state the theorems about the *privacy* and *local verifiability*, as well as giving a brief high-level overview of the proofs. The full proofs are presented in the following two subsections.

Theorem 1. *This construction is* private *according to Definition 2 under the assumption that the RSA problem is hard.*

The proof of this theorem proceeds in the following steps: it can be seen that the scheme is secure if the circuit contains no fanout gates, since in this case (roughly speaking) the adversary is given $(x^\tau, \mathsf{Ext}(x) \oplus m)$ and is asked to output some information about m. Any such adversary can be used to break the RSA function x^τ since Ext extracts the hard-core bits. To deal with the fanout gates we construct a series of hybrids where at each step we decompose the circuit C into a circuit C^* which does not contain any fanout gate and a circuit C' which takes two extra inputs (both set to be the output of C^* and one fanout gate less than the original circuit C. It is possible to argue that an adversary which succeeds in breaking the security for the original circuit C can be used to break the security of the scheme run on the circuit C' with one less fanout gate, roughly thanks to the security of the encryption scheme used in the construction of the fanout gates and the fact that the rest of the circuit C^* does not contain any fanout gate.

Theorem 2. *This construction is* locally verifiable *according to Definition 3.*

The proof of this theorem proceeds by first noting that the tag function is a permutation and hence the set of tags uniquely defines a set of numbers mod N that are supposed to act as the "wire values" w_i in the circuit C. One then checks that the verification ensures that the values on wires going into a gate correctly correspond to the value on the output wire. For fanout gates this check crucially relies on the observation above (1), that the tag function commutes with the encryption scheme. Hence, if everything checks out, the public data must correspond to a correct execution of the sharing algorithm, and therefore by the correctness property all qualified sets will reconstruct the same secret.

4.3 Proof of Privacy (Theorem 1)

Proof. We construct the simulator Sim as following:

1. Sim samples $r \leftarrow \{0,1\}^\mu$;
2. Sim runs $(s_0, s_1, \ldots, s_n) \leftarrow \mathsf{Share}_{pp}(C, r)$
3. Sim outputs $(s_0, \{s_i\}_{i \in I})$.

We now prove that the output of Sim is computationally indistinguishable from the distribution of the real output.

From Decision to Search. Since τ is a prime number larger than N the function Tag is a one-way permutation. The only difference between a the real view and the simulated view is that in the real view $u = \mathsf{Ext}(w_\ell) \oplus m$ whereas in the simulated view u is a uniformly random value. Since Ext extracts the hard core bits of the RSA function Tag, any distinguisher $\mathcal{D}$ that distinguishes between the real view and the simulated view can be turned into an algorithm $\mathcal{D}'$ that outputs w_ℓ. We now prove that computing w_ℓ is impossible without breaking the RSA assumption.

Without Fanout Gates. We start by noting that if the circuit C had no fanout gates, then it is computationally hard to find the value w_ℓ: Without fanout gates the value corresponding to the output wire w_ℓ is simply the product of a subset of the input wires w_i i.e., $w_\ell = \prod_{i \in S} s_i$ for some set S (note the set may not be unique, in which case we consider the first such set in lexicographical order). In the privacy game the adversary only sees shares for an unqualified set i.e., the adversary receives $\{s_i\}_{i \in I}$ for some I such that $C(\mathsf{set2bits}(I)) = 0$, meaning that $T = S \setminus I \neq \emptyset$. Now an adversary who computes w_ℓ can be turned into an adversary who computes $y = \prod_{i \in T} s_i$, which is equivalent to breaking the one-way property of the permutation Tag. In particular, since Tag is homomorphic, given such an adversary a reduction can solve an RSA instance $(\tau, y = x^\tau)$ by choosing $|T| - 1$ random s_i's, defining the last s_i such that $y = \prod_{i \in T} s_i$ and computing the respective tags. Note that the same argument can be made for any internal wire: let $C_i(x)$ be the circuit which outputs the same value as gate g_i in C, then if $C_i(\mathsf{set2bits}(I)) = 0$ then no adversary can output the value w_{i_O}.

Removing Fanout Gates. The core of the proof is to show how we can "get rid of" the fanout gates by decomposing C into circuits without any fanout gates. We proceed as follows: $C^*_{\ell_{\mathsf{fanout}}}$ is a circuit which takes an input of length $n^*_{\ell_{\mathsf{fanout}}} = n + 2\ell_{\mathsf{fanout}}$ wires (the input wires plus all the output wires of the fanout gates) and contains all gates that can be reached from the output wire without traversing any fanout gate. As the next step, we take the input wire of the first fanout gate encountered in the previous process, and we define $C^*_{\ell_{\mathsf{fanout}}-1}$ as the circuit which takes an input of length $n^*_{\ell_{\mathsf{fanout}}-1} = n + 2(\ell_{\mathsf{fanout}} - 1)$ and contains all gates that can be reached from this wire without traversing any fanout gate. The process stop with C^*_0 which is guaranteed to take an input of length at most $n^*_0 = n$, that is its input is the same as the original circuit C and in particular none of its inputs come from the output of any fanout gates.

We now define the values $x_{n+1} = x_{n+2} = C^*_0(x_1, \dots, x_n)$, $x_{n+3} = x_{n+4} = C^*_1(x_1, \dots, x_{n^*_1})$ up to $x_{n+2\ell_{\mathsf{fanout}}-1} = x_{n+2\ell_{\mathsf{fanout}}} = C^*_{\ell_{\mathsf{fanout}}-1}(x_1, \dots, x_{n^*_{\ell_{\mathsf{fanout}}-1}})$. It is convenient to define, given a set I such that $x = \mathsf{set2bits}(I)$, a set I^* such that $\mathsf{set2bits}(I^*) = (x_1, \dots, x_{n+2\ell_{\mathsf{fanout}}})$. It is now clear by inspection that

$$C(x) = C^*_{\ell_{\mathsf{fanout}}}(x_1, \dots, x_{n^*_{\ell_{\mathsf{fanout}}}})\ \forall x$$

Crucially all the circuits $(C^*_0, \dots, C^*_{\ell_{\mathsf{fanout}}})$ do not contain any fanout gates. We construct also circuits $(C'_0, \dots, C'_{\ell_{\mathsf{fanout}}})$, where C'_j takes an input of size at most $n'_j = n + 2j$ and has exactly $\ell_{\mathsf{fanout}} - j$ fanout gates. We define $C'_{\ell_{\mathsf{fanout}}} = C^*_{\ell_{\mathsf{fanout}}}$ and C'_{j-1} to be equal to

$$C'_j(x_1, \dots, x_{n'_{j-1}}, C^*_j(x_1, \dots, x_{n'_{j-1}}), C^*_j(x_1, \dots, x_{n'_{j-1}}))$$

It is clear by inspection that $C = C'_0$.

After the heavy but necessary notation, we are ready for showing our reduction. We construct a series of adversaries $\mathcal{D}_j$ who get as input

$$(s_0, s_1, \dots, s_{n+2j}) \leftarrow \mathsf{Share}_{pp}(C'_j, r)$$

We have already argued that no adversary can output the value corresponding to the output wire (which is necessary to distinguish between the real and simulated execution) if the circuit does not contain any fanout gates, which implies that $\mathcal{D}_{\ell_{\mathsf{fanout}}}$ cannot either without breaking the RSA assumption. We then show that if $\mathcal{D}_{j-1}$ succeeds in outputting the value of the output wire for C'_{j-1} with noticeable probability then we can construct $\mathcal{D}_j$ who outputs the value of the output wire for C'_j as well. Using standard hybrid arguments we can therefore conclude that the adversary $\mathcal{D} = \mathcal{D}_0$ can only succeed in breaking privacy by breaking the RSA assumption.

The reduction goes as follows: $\mathcal{D}_j$ gets as input all the shares s'_i for all $i \in I^* \cap [n + 2j]$, where $(s'_0, s'_1, \dots, s'_{n+2\cdot j}) \leftarrow \mathsf{Share}_{pp}(C'_j, r)$. Remember that $C(\mathsf{set2bits}(I)) = 0$.

Now the $\mathcal{D}_j$ needs to construct a sharing for the circuit C'_{j-1} which is of the format expected by $\mathcal{D}_{j-1}$. Intuitively this is done by adding a single fanout gate to the circuit and running the share procedure for the circuit C^*_j. The complication here is that the shares $(s^*_0, s^*_1, \dots, s^*_{n+2j-2})$ for the circuit C^*_j must be consistent with the existing shares for all known shares i.e., it must be that $s^*_i = s'_i\ i \in I^* \cap [n + 2j]$.

Case 1: *(the output of C^*_j is 1)*

The easier case is when $C^*_j(\mathsf{set2bits}(I^* \cap [n + 2j - 2])) = 1$ since in this case no values associated with the fanout gate we are introducing are supposed to stay hidden from $\mathcal{D}_{j-1}$ – in this case $\mathcal{D}_j$ runs the reconstruction procedure for C^*_j using the known shares and, since C^* does not contain any fanout gate, the reconstruction boils down to multiplying the shares for any qualified set S_j for C^*_j i.e., $k_j = \prod_{i \in S_j} s'_i$ and compute encryptions $c_{i_{LO}} \leftarrow \mathsf{E}(k_j, s'_{n+2j-1}), c_{i_{RO}} \leftarrow \mathsf{E}(k_j, s'_{n+2j})$. Finally $\mathcal{D}_j$ sets $s^*_0 = s'_0||(c_{i_{LO}}, c_{i_{RO}})$ and $s^*_i = s'_i$ and gives these values as input to $\mathcal{D}_{j-1}$.

Now $\mathcal{D}_j$ simply outputs whatever $\mathcal{D}_{j-1}$ outputs and wins the game: In this case the values received by $\mathcal{D}_{j-1}$ are distributed exactly as in the real experiment, and therefore the probability with which $\mathcal{D}_{j-1}$ will output the value for the output wire is exactly exactly the same.

Case 2: *(the output of C_j^* is 0, but s'_{n+2j-1} and s'_{n+2j} are known anyway)*
This case happens if, for example, both outputs of the fanout gate are input to OR gates which evaluate to 1. Since in the construction we set both input values of an OR gate to be equal to its output, this means that both $\mathcal{D}_j$ and $\mathcal{D}_{j-1}$ know these values. In this case $\mathcal{D}_j$ will simply choose a random k and encrypt those shares as above. Since $C_j^*(\mathsf{set2bits}(I^* \cap [n+2j-2])) = 0$ and C_j^* does not contain any fanout gate, this implies that $\mathcal{D}_j$ cannot tell the difference without breaking the RSA assumption. Note that in this case we are *not* relying on the security of the encryption scheme – having access to both encrypted output $\mathcal{D}_j$ can actually recover k. The point here is that since $k = \prod_{i \in S_j} s'_i$ and the adversary does not know at least one of these s'_i's, we can conclude that an adversary which distinguishes successfully can be used to break the one-wayness of of Tag.

Case 3: *(the output of C_j^* is 0 and at least one between s'_{n+2j-1} and s'_{n+2j} is not known)*
The case where $C_j^*(\mathsf{set2bits}(I^* \cap [n+2j-2])) = 1$ and at least one between s'_{n+2j-1} and s'_{n+2j} is not known is the most interesting, since in this case we rely on the security of the encryption scheme. Wlog say that $\mathcal{D}_{j-1}$ knows s'_{n+2j-1} but not s'_{n+2j} (the case where both are unknown follow in a straightforward way): now $\mathcal{D}_{j-1}$ invokes the *one-query CPA oracle* for the encryption scheme with s'_{n+2j-1} and receives $c_{i_{LO}} \leftarrow \mathsf{E}(k, s'_{n+2j-1})$ and $c_{i_{RO}} \leftarrow \mathsf{E}(k, r)$, and if $\mathcal{D}_j$ outputs the value of the output wire $\mathcal{D}_{j-1}$ can reconstruct r and therefore break the security of the encryption scheme.

4.4 Proof of Verifiability (Theorem 2)

Proof. To prove local verifiability we first observe that since Tag is a permutation, given any y there exist a single x such that $\mathsf{Tag}(y) = x$.

Given any qualified set $I \subseteq [n]$ such that $C(\mathsf{set2bits}(I)) = 1$ we define a set $I' \subseteq [\ell]$ such that $i \in I'$ if $w_i = 1$ during the (plain, Boolean) evaluation of $C(\mathsf{set2bits}(I))$. Let w_i^V the values assigned to the wires during the verification phase, and w_i^R the values assigned to the wires during the reconstruction phase using the (qualified) set I. We prove that if $\mathsf{Ver}_{pp}(s_0, s_i) = \mathtt{true}$ for all $i \in [n]$, then for all $i \in I'$ it holds that $\mathsf{Tag}(w_i^R) = w_i^V$.

We prove this by induction. Thanks to Step 2 in Ver, it holds that

$$\mathsf{Tag}(w_i^R) = w_i^V \text{ for all } i \in I \subset I'$$

Now take the next wire $i \in I' \setminus I$ and wlog assume that i is the output of a gate g_j that only takes inputs from wires with index in $[n+i] \cap I'$ (one can always reorder the wires to make sure that this happens). We can now argue that:

- If g_j is an AND gate, then the value on wire $i = j_O$ is a function of the values on wires j_{LI} and j_{RI}, and since $i \in I'$ it must also be the case that j_{LI} and j_{RI} are in $I' \cap [n+i]$ (the output of the AND gate is set only if both input wires are set), which allows to use the induction hypothesis. By induction it holds that $\mathsf{Tag}(w_{j_{LI}}^R) = w_{j_{LI}}^V$ and $\mathsf{Tag}(w_{j_{RI}}^R) = w_{j_{RI}}^V$. Now since $w_{j_O}^R = w_{j_{LI}}^R \cdot w_{j_{RI}}^R$ and also $w_{j_O}^V = w_{j_{LI}}^V \cdot w_{j_{RI}}^V$ and, since the Tag function is homomorphic this implies that:
$$\mathsf{Tag}(w_{j_O}^R) = \mathsf{Tag}(w_{j_{LI}}^R \cdot w_{j_{RI}}^R) = w_{j_{LI}}^V \cdot w_{j_{RI}}^V = w_{j_O}^V$$
- If g_j is an OR gate, then the value on wire $i = j_O$ is a function of the values on wires j_{LI} and j_{RI}, and since $i \in I'$ it must also be the case that at least one between j_{LI} and j_{RI} are in $I' \cap [n+i]$ (the output of the OR gate is set only if at least one of the input wires are set), which allows to use the induction hypothesis: by induction it holds that at least one between $\mathsf{Tag}(w_{j_{LI}}^R) = w_{j_{LI}}^V$ and $\mathsf{Tag}(w_{j_{RI}}^R) = w_{j_{RI}}^V$ holds. During the verification phase $w_{j_O}^V = w_{j_{LI}}^V$ only if $w_{j_{LI}}^V = w_{j_{RI}}^V$, instead during the construction phase $w_{j_O}^R$ might be set to $w_{j_{LI}}^R$ or $w_{j_{LI}}^R$ depending on which qualified set is being used. But since Tag is a permutation, $\mathsf{Tag}(a) = \mathsf{Tag}(b)$ implies that $a = b$, and therefore during the reconstruction we have that if $w_{j_{LI}}^R \neq \perp$ and $w_{j_{RI}}^R \neq \perp$ then $w_{j_{LI}}^R = w_{j_{RI}}^R$ and therefore
$$\mathsf{Tag}(w_{j_O}^R) = \mathsf{Tag}(w_{j_{LI}}^R) = w_{j_{LI}}^V = w_{j_O}^V$$

– If g_j is a fanout gate, then the value on the wire $i = j_{LO}$ (the case where $i = j_{RO}$ can be argued exactly in the same way) is a function of the value on the wire j_I and in the public share s_0. During the verification phase $w^V_{j_{LO}} = t_{j_{LO}}$ only if $t_{j_{LO}} = \mathsf{D}(w^V_{j_I}, \rho_{j_{LO}}, \mathsf{Tag}(\sigma_{j_{LO}}))$ while in the reconstruction phase $w^R_{j_{LO}} = \mathsf{D}(w^R_{j_I}, c_{j_{LO}})$. By induction it holds that $\mathsf{Tag}(w^R_{j_I}) = w^V_{j_I}$. Using the fact that the encryption scheme and the Tag function commute we show that:

$$\begin{aligned}\mathsf{Tag}(w^R_{j_{LO}}) &= \mathsf{D}(w^R_{j_I}, c_{j_{LO}})^\tau = \mathsf{D}((w^R_{j_I})^\tau, \rho_{j_{LO}}, \sigma^\tau_{j_{LO}}) \\ &= \mathsf{D}(w^V_{j_I}, \rho_{j_{LO}}, \mathsf{Tag}(\sigma_{j_LO})) = w^V_{j_{LO}}\end{aligned}$$

Since $\ell \in I'$ (the set is qualified), this finally implies that $\mathsf{Tag}(w^R_\ell) = w^V_\ell$, and thanks to Step 5 in the Ver algorithm we can conclude that $w^V_\ell = t_\ell$ and therefore the value w^R_ℓ is the same for all qualified sets. Finally, since $m = u \oplus \mathsf{Ext}(w^R_\ell)$ is a deterministic function of u and w^R_ℓ, we can conclude that all qualified sets reconstruct the same secret m.

5 Globally Verifiable Secret Sharing Schemes

In the previous section we have presented a scheme where each player can check whether the private share received from the dealer is consistent with the public share. In this section we present two possible ways of implementing private channels from the dealer to the players, which also allow to deal with the case where the dealer is cheating and the honest players need to reach an agreement.

In both extensions, we let the setup algorithm output some additional parameters and decryption keys for all players. Now the dealer, instead of sending shares privately to each player, appends encryptions of the shares to the tags of the shares which are sent over the broadcast channel, and each player can recover her own share.

In the first proposal, we use an encryption scheme with the property that it is possible to verify, given an encryption and a tag, whether they contain the same value. This is done using efficient non-interactive zero-knowledge proofs by compiling efficient sigma-protocols for the statement using the Fiat-Shamir heuristic. Doing so makes the scheme *non-interactive* and *publicly-verifiable*, since everyone can check that the dealer sent correct shares to all players. In the second proposal, we let the dealer encrypt the shares using an encryption scheme which has a special property, namely it allows the receiver to prove to a third party what has been received: this scheme gets rid of the random oracle model, but requires each (complaining) party to send a single message to all other parties using a broadcast channel.

Notation. We need to redefine the syntax and the functioning of the setup phase and the sharing phase (the reconstruction phase is unchanged and the syntax of the verification phase differs for the two schemes). The scheme uses a public key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Setup: The setup algorithm for the *globally-verifiable* CS3 outputs

$$(pp, \{d_i\}_{i\in[n]}) \leftarrow \mathsf{gvSetup}(1^\kappa)$$

where d_i is the decryption key for P_i (the corresponding encryption key for P_i can be derived from (pp, i)). The decryption keys are sent to the owner using private channels, where the public parameters are made public; The algorithm $\mathsf{gvSetup}(1^\kappa)$ simply runs $pp' \leftarrow \mathsf{Setup}(1^\kappa)$ to generate the public parameters for our underlying locally verifiable scheme and n copies of $\mathsf{Gen}_{pp}(1^\kappa)$ to generate n encryption/decryption key pairs (e_i, d_i), and finally outputs $pp = (pp', e_1, \ldots, e_n)$.

Share: The share algorithm for the *globally-verifiable* CS3 outputs

$$s \leftarrow \mathsf{gvShare}_{pp}(C, m)$$

and the dealer broadcasts s; The algorithm $\mathsf{gvShare}_{pp}(C, m)$ simply runs $\mathsf{Share}_{pp}(C, m)$ to generate $(s_0, s_1, \ldots, s_n)$, generates $z_i \leftarrow \mathsf{Enc}_{pp}(e_i, s_i)$ for all $i \in [n]$ and outputs $s = (s_0, z_1, \ldots, z_n)$.

We need also to redefine correctness and privacy as follows:

Definition 4 (Correctness). *A globally verifiable CS3 π is* correct *if for all $m \in \mathcal{M}$, for all circuits C describing an access structure $\mathcal{A}$, and for all $I \in \mathcal{A}$,*

$$\Pr[m \neq \mathsf{Rec}_{pp}(s, \{d_i\}_{i \in I})] \leq \mathsf{negl}(\kappa)$$

where $s \leftarrow \mathsf{gvShare}_{pp}(C, m), (pp, \{d_i\}_{i \in [n]}) \leftarrow \mathsf{gvSetup}(1^\kappa)$ and the probabilities are taken over the choices of all algorithms.

It is trivial to see that combining a correct locally verifiable CS3 with an encryption scheme with a correct decryption leads to a correct globally verifiable CS3.

Definition 5 (Privacy). *A globally verifiable CS3 π is* private *if for all circuits C describing an access structure $\mathcal{A}$, and for all $I \notin \mathcal{A}$, there exist a PPT simulator Sim such that for all $m \in \mathcal{M}$,*

$$\mathsf{Sim}(pp, C, I) \approx_c (s, \{d_i\}_{i \in I})$$

where $(pp, \{d_i\}_{i \in [n]}) \leftarrow \mathsf{gvSetup}(1^\kappa)$, $s \leftarrow \mathsf{gvShare}_{pp}(C, m)$.

It is trivial to see that combining a private locally verifiable CS3 with a semantically secure encryption scheme leads to a private globally verifiable CS3.

5.1 Non-Interactive and Publicly-Verifiable Scheme

Our first proposal is a non-interactive and publicly-verifiable VCS3. The syntax of the verification scheme here is:

Verification: The verification algorithm $f \leftarrow \mathsf{niVer}_{pp}(s)$ outputs a bit $f \in \{\texttt{true}, \texttt{false}\}$ which indicates whether s is a valid sharing or not. Note that anyone can run the verification phase i.e., one does not need to know any of the decryption keys d_i to run this algorithm.

The scheme should satisfy the following property:

Definition 6 (Public-Verifiability). *We say a CS3 scheme is* publicly verifiable *if for all $n \in \mathbb{Z}$, for all circuits C describing an access structure, and for all PPT algorithms D^* the following holds: Let $(pp, d_1, \ldots, d_n) \leftarrow \mathsf{Setup}(1^\kappa)$ and $s \leftarrow D^*(pp)$. If $\mathsf{niVer}_{pp}(s) = \texttt{true}$, then with a overwhelming probability there exists a value $m \in \mathcal{M}$ such that $\mathsf{Rec}_{pp}(s, \{d_i\}_{i \in I}) = m$ for all qualified sets $I \in \mathcal{A}$ (where $\mathcal{A}$ is defined in s).*

We construct the verification algorithm for this scheme, niVer, by replacing Step 2 in Ver (defined in Section 4.2) with the following:

3. If $\exists i \in [n] s.t., \mathsf{Tag}(\mathsf{Dec}_{pp}(d_i, e_i)) \neq t_i$ stop and output $\texttt{false}$; else:

This condition can be checked efficiently using the NIZKs. Soundness of the NIZKs together with the *local verifiability* of the underlying scheme implies *public verifiability.*

We finally describe how we construct the encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and the NIZKs π_i. The scheme is essentially ElGamal encryption in the RSA group. We choose $N = pq$ where $p = 2p' + 1, q = 2q' + 1$ for primes p', q'. This ensures that the subgroup G of numbers with Jacobi symbol 1 mod N is cyclic of order $2p'q'$, and we let g be a generator of G. The encryption scheme we will construct is secure when applied to messages in G. We therefore need to slightly change the VSS constructed above so that wire values and tags are chosen from G and not from all of $\mathbb{Z}_N^*$. Since Jacobi symbols can be computed efficiently, one can always check that the dealer chooses his values correctly. The encryption scheme now works as follows:

Generation: In the set-up we sample a random decryption key $d_i \in \mathbb{Z}_N$ and output the corresponding encryption key $e_i = g^{d_i}$;

Encryption: sample a random $r \in \mathbb{Z}_N$ and output $z_i = (\alpha_i, \beta_i) = (g^r, {e_i}^r \cdot s_i)$;
Decryption: output $s_i = \beta_i \cdot \alpha_i^{-d_i}$;

We now need to construct a NIZK that allows a prover with witness d_i to persuade a verifier who knows $(\tau, e_i, t_i, \alpha_i, \beta_i)$ that

$$(g, e_i, \alpha_i^\tau, \beta_i^\tau \cdot t_i^{-1})$$

is a DDH tuple: note that when the dealer is honest this is indeed the case, i.e., the tuple in question is:

$$(g, g^{d_i}, g^{r\cdot\tau}, g^{d_i \cdot r \cdot \tau} \cdot (s_i^\tau / s_i^\tau)) = (g, g^{d_i}, g^{r\cdot\tau}, g^{(r\cdot\tau)\cdot d_i})$$

Very efficient sigma-protocols for this language are well-known (see e.g. [HL10]), which can be made non-interactive in the random oracle model using the Fiat-Shamir Heuristic.

5.2 Non-Interactive Scheme (with Complaints)

The main disadvantage of the previous solution is that it requires the random oracle model for the NIZKs. Our second proposal instead uses one-round complaints to ensure verifiability. The idea here is that every player retrieves her share from the public encryptions, and if the share does not match the public tag, the player can complain by broadcasting some information that allows all other parties to check that the dealer cheated. In particular we do not wish to allow a corrupt player to unfairly accuse an honest dealer, and the "proof of cheating" should not disclose any other information. Both these properties can be achieved using a technique introduced in [DT07] and formalized in [DHKT08]. We refer to [DHKT08] for the details of this method, and we only sketch the high-level idea here: The idea is to let the dealer encrypt the shares using an *identity-based encryption scheme with verifiable secret keys (IBE-VSK)*: in this setting the decryption key of each player is the master secret key for the IBE scheme (and the encryption key is the corresponding public key). When the dealer encrypts the shares for all the parties, he does so using a unique *id* as the identity in the IBE scheme. Each player can decrypt by generating the secret key sk_{id} corresponding to this *id* and then perform the IBE decryption. To complain, the player can broadcast the secret key sk_{id}, and all other parties are now able to retrieve the share and check whether it is consistent with the tag. The security of the IBE scheme implies that revealing sk_{id} does not disclose any information about the encrypted shares in other sessions. In addition, the VSK property allows all other players to verify that sk_{id} is indeed the secret key corresponding to the *id* for the public key of that player, and was not maliciously generated to accuse an honest dealer. We note that VSK is a mild assumption, and every proposed efficient IBE satisfies the VSK property [DT07,DHKT08].

References

ACGS88. Werner Alexi, Benny Chor, Oded Goldreich, and Claus P Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, 1988.

BL90. Josh Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *Proceedings on Advances in cryptology*, pages 27–35. Springer-Verlag New York, Inc., 1990.

Bla79. George Robert Blakley. Safeguarding cryptographic keys. In *National Computer Conference*, pages 313–317. IEEE Computer Society, 1979.

CDD00. Ronald Cramer, Ivan Damgård, and Stefan Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 325–334. ACM, 2000.

CDM00. Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology–EUROCRYPT 2000*, pages 316–334. Springer, 2000.

DHKT08. Ivan Damgård, Dennis Hofheinz, Eike Kiltz, and Rune Thorbek. Public-key encryption with non-interactive opening. In *Topics in Cryptology–CT-RSA 2008*, pages 239–255. Springer, 2008.

DT07. Ivan Damgård and Rune Thorbek. Non-interactive proofs for integer multiplication. In *Advances in Cryptology–EUROCRYPT 2007*, pages 412–429. Springer, 2007.

HL10. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer, 2010.

ISN89. Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

KW93. Mauricio Karchmer and Avi Wigderson. On span programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993.

Ped92. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology–CRYPTO91*, pages 129–140. Springer, 1992.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Sta96. Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology–EUROCRYPT96*, pages 190–199. Springer, 1996.

VNS$^+$03. V. Vinod, Arvind Narayanan, K. Srinathan, C. Pandu Rangan, and Kwangjo Kim. On the power of computational secret sharing. In *Progress in Cryptology – INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, pages 162–176, 2003.

Yao89. Andrew C. Yao. Unpublished manuscript. *Presented at Oberwolfach and DIMACS workshops.*, 1989.