Linear Integer Secret Sharing and Distributed Exponentiation

Ivan Damgård[⋆] and Rune Thorbek

BRICS[⋆⋆], Dept. of Computer Science, University of Aarhus

Abstract. We introduce the notion of Linear Integer Secret-Sharing (LISS) schemes, and show constructions of such schemes for any access structure. We show that any LISS scheme can be used to build a secure distributed protocol for exponentiation in any group. This implies, for instance, distributed RSA protocols for arbitrary access structures and with arbitrary public exponents.

1 Introduction

In a secret sharing scheme, a *dealer* distributes *shares* of a secret to a number of shareholders, such that only certain designated subsets of them - the *qualified sets* can reconstruct the secret, while other subsets have no information about it. The collection of qualified sets is called the *access structure*. In particular, the access structure consisting of all sets of cardinality greater than t is called a *threshold-t* structure.

Secret Sharing was first introduced[20] as a way to store critical information such that we get at the same time protection of privacy and security against loosing the information. Later, secret sharing has proved extremely useful, not just as a passive storage mechanism, but also as a tool in interactive protocols, for instance in threshold cryptography. Here, the private key in a public key scheme is secret shared among a set of servers, and the idea is that a qualified subset of the servers can use their shares to help a client to decrypt or sign an input message, but without having to reconstruct the private key in a single location. As long as an adversary cannot corrupt too large a subset of the servers, he cannot prevent the system from working, nor can he learn any information on the private key.

The central operation we need to perform securely in these applications is typically an exponentiation, that is, we are given some finite group G and an input $a \in G$, and we want to compute a^s, where s is a secret exponent which has been secret-shared among the servers. In some cases the group order is a public prime q. The problem is then straightforward to solve since we can use any standard linear secret sharing scheme over the field Z_q. The observation is

[⋆] FICS, Foundations in Cryptography and Security, center supported by the Danish research Council

[⋆⋆] Basic Research in Computer Science, Center of the Danish National research Foundation simply that for any linear scheme (such as Shamir's) over Z_q, the secret can be written as a linear combination $s = \sum_{i \in I} \alpha_i s_i \bmod q$, where I is any qualified set of servers holding shares $\{s_i | \ i \in I\}$, and where the α_i's can be computed from the index set I. Now, if the servers provide $a_i = a^{s_i}$ (and prove they did so correctly), we can compute $a^s = \prod_{i \in I} a_i^{\alpha_i}$. However, there are other cases where the group order is not prime and is not public (or even unknown to everyone), such as when G is Z_N^* for an RSA modulus N or when G is a class group. This leads to various problems: it would be natural to try to build a secret sharing scheme over Z_t where t is the order of G, but the standard constructions do not immediately work if t is not a prime. Matters are of course even worse if t is unknown to everyone.

The literature contains many techniques for getting around these problems. The techniques work in various particular scenarios, but they all have shortcomings in general. We give a short overview here:

- The black-box secret sharing schemes of [8, 13, 21] can be used to share a secret chosen from any Abelian group, including Z_t. This requires, of course, that the dealer knows t so he can do computations in Z_t. This is never the case if G is a class group, and if $G = Z_N^*$, the dealer must know the factorization of N. Note that in proactive threshold RSA schemes, each player typically has to reshare his share of the private key from time to time, however, we can of course not afford to reveal the factorization of N to every shareholder.
- In Shoup's threshold RSA protocol[22], the idea is to restrict the modulus N to be a safe prime product, which allows us to work in a subgroup of Z_N^* whose order is the product of two large primes. This is "close enough" to a prime so that standard Shamir sharing of s will work. This requires that the dealer knows the factorization. Moreover, for technical reasons, the protocol can only compute $a^{s \cdot n!}$ where n is the number of servers. This is solved by exploiting that we have the public exponent e available. Assuming e is relatively prime to $n!$, we can compute a^s efficiently. The problem in general is of course that we may not always be able to choose the group order as we like, and the inverse of s modulo the group order may not always be available or it may not be prime to $n!$. For instance, we cannot use small public exponents such as 3.[1]
- The secret sharing scheme of [15] which was also used in [12, 10] is a variant of Shamir's scheme, where we use polynomials over the integers. Using this to share s does not require any knowledge of the order of G. However, the scheme does not allow reconstruction of s by a linear combination of shares, instead one obtains the secret times some constant, typically $s \cdot n!$. This causes the protocol to produce $a^{s \cdot n!}$ as output, and we have the same problem as with Shoup's protocol.

[1] Shoup suggests an alternative solution where any public exponent can be used, but this requires that one additionally assumes that the DDH assumption holds in the RSA group.

– Finally, the method of Rabin [18] uses secret sharing in "two levels", i.e., the secret exponent s is shared additively, such that $s = s_1 + ... + s_n$ where server i knows s_i, and then s_i is itself secret shared among the servers. Schemes of this type require no knowledge of the group order to do the sharing since in principle, any secret-sharing scheme can be used to share the s_i's. On the other hand, shares become larger than with other schemes and extra rounds of interaction is needed (to reconstruct s_i) as soon as even one server i fails to participate correctly. Hence (in contrast to the other types of protocols) this approach cannot be made non-interactive, not even in the random oracle model.

A final issue with current state of the art of distributed exponentiation is that known solutions (except the two-level method) do not generalize to non-threshold access structures. The point of general structures is that when we secret share the private key according to a threshold structure, we are implicitly assuming that all servers are equally easy to break into, and so the only important parameter is the *number* of corrupted servers. In reality, some servers may well be more reliable than others, and so we may need to specify which sets should be qualified in a more flexible way, that is, we need a more general access structure.

1.1 Our Results

In this paper, we introduce a type of secret sharing scheme called *Linear Integer Secret-Sharing* (LISS). In a LISS scheme, the secret is an integer chosen from a (publically known) interval, and each share is computed as an integer linear combination of the secret and some random numbers chosen by the dealer. Reconstruction of the secret is also by computing a linear combination with integer coefficients of the shares in a qualified set.

LISS schemes are closely related to - but not the same as - the black-box secret sharing schemes (BBSS) mentioned earlier of Desmedt-Frankl[13] and Cramer-Fehr[8]. Whereas BBSS schemes are designed to secret share elements from any *finite* abelian group and use computations in this group to do it, our computations are done over the (infinite) ring of integers. This difference has a number of consequences that we return to below. LISS schemes are also different from the method in [15] based on integer polynomials, since they require a final division to get the secret while for LISS schemes we insist that linear combinations be sufficient.

Note that it was shown in [5, 6] that perfect secret sharing and private computation over countably infinite domains (like the integers) is not possible. However, this does not rule out schemes of our type since we restrict our secrets to be chosen from a publically known interval and only aim for statistical rather than perfect privacy.

Cramer and Fehr introduce the concept of an integer span program (ISP) and use it to construct BBSS schemes. We show that any ISP can also be used to build a secure LISS scheme. Roughly speaking, an ISP is specified by a matrix with integer entries, and these entries are used as coefficients in the linear combinations that produce the shares from secret and randomness. In particular, the construction from [8] of an ISP for threshold-t access structures implies a LISS scheme for the same structure. Moreover, we revisit the well known construction of Benaloh and Leichter [1] based on monotone formulas that was originally conceived for a finite Abelian group, and we show that a LISS scheme can be built from any monotone formula. This implies that a LISS schemes exists for any access structure, though not necessarily an efficient one.

The ISP construction of Cramer and Fehr was shown to imply optimal threshold BBSS schemes. We show that this is not always the case for LISS schemes: if we base the Benaloh-Leichter construction on a monotone formula for the threshold function, we obtain threshold LISS schemes. It now turns out that, depending on how small a formula we can produce, this construction may produce a threshold LISS scheme with smaller shares or smaller randomness complexity than those coming from the Cramer-Fehr construction. With current of state of the art, this does not happen in general, but we find that for a fixed threshold and a large number of players, there are monotone formula constructions that produce smaller shares than Cramer-Fehr[2].

It is interesting to note that if the known lower bound on the montone formula size for the threshold function [3] turn out to be tight, this would make the Benaloh-Leichter construction more efficient in general than the Cramer-Fehr construction. While this may not seem likely with our current knowledge, it does mean that determining the efficiency of an optimal threshold LISS scheme remains an open question. The reason why BBSS schemes are different from LISS schemes in this respect is that when we use an ISP for building a BBSS scheme, the size of shares we get is independent of the size of the integers occurring in the description of the ISP, but this is no longer true when we build a LISS scheme.

Finally, we show that any LISS scheme can be used to build a distributed exponentiation protocol. The protocol does not use multilevel secret sharing. Thus, it can be made non-interactive using any of the known techniques for this purpose, such as the Fiat-Shamir heuristic (the random oracle model) or [7, 11, 16]. Furthermore, no player, including the dealer, needs to know the order of the group involved. This implies that we obtain the first non-interactive distributed exponentiation protocol that works for any group and any access structure.

We also look at the particular case of distributed RSA. We generalize the results of Damgård and Dupont[10] to arbitrary access structures, and thus obtain a distributed RSA signature scheme for any access structure, any public exponent and any modulus, efficiently and in constant-round without using random oracles or any assumptions other than the RSA assumption.

We emphasize that our result that all LISS schemes can be used for distributed exponentiation does not hold for BBSS schemes, not even if we assume

[2] Note that in a later paper[9], Cramer, Fehr and Stam propose a construction that they conjecture to be more efficient than[8], but so far, the asymptotic efficiency of the scheme remains unproved.

that the dealer knows the group order[3]. The reason for this is that in order to do the proof of security for an exponentiation protocol using known simulation techniques, the secret sharing scheme needs to have the so called *share completion property:* given an unqualified set of shares and the secret, we can compute by linear combinations a *complete* set of shares consistent with what we were given. It is not known whether BBSS or LISS schemes have this property in general, in fact the answer is probably no. Here, we get around this problem by coming up with a different simulation technique where share completion is not needed. This technique always works with a LISS scheme, but fails with BBSS when the group order is not public.

2 Linear Integer Secret Sharing

First we formally define the required access structures.

Definition 1. *A* monotone access structure *on* $\{1,\ldots,n\}$ *is a non-empty collection* Γ *of sets* $A \subseteq \{1,\ldots,n\}$ *such that* $\emptyset \notin \Gamma$ *and such that for all* $A \in \Gamma$ *and for all sets* B *with* $A \subseteq B \subseteq \{1,\ldots,n\}$ *it holds that* $B \in \Gamma$.

Definition 2. *Let* t *and* n *be integers with* $0 < t < n$. *The* threshold-t access structure $T_{t,n}$ *is the collection of sets* $A \subseteq \{1,\ldots,n\}$ *with* $|A| > t$.

Let $P = \{1,\ldots,n\}$ denote the n shareholders (or players) and D the dealer. Let Γ be a monotone access structure on P. The dealer D wants to share a secret s from the publically known interval $[0..2^l]$ to the shareholders P over Γ, such that every set of shareholders $A \in \Gamma$ can reconstruct s, but such that a set of shareholders $A \notin \Gamma$ get no or little information on s. We call the sets which are allowed to reconstruct the secret *qualified* and the sets which should not be able to obtain any information about the secret *forbidden*.

For this purpose we use a *distribution matrix* $M \in Z^{d\times e}$ and a *distribution vector* $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$, where s is the secret, and the ρ_i's are uniformly random chosen integers in $[0..2^{l_0+k}]$ for $2 \le i \le e$, where k is the security parameter and l_0 is a constant that is part of the description of the scheme. The dealer D calculates shares by

$$M \cdot \boldsymbol{\rho} = (s_1, \ldots, s_d)^T, \tag{1}$$

where we denote each s_i as a *share unit* for $1 \le i \le d$. Let $\psi : \{1,\ldots,d\} \to P$ be a surjective function. The i'th share unit is then given to the $\psi(i)$'th shareholder, we say that $\psi(i)$ owns the i'th row in M. If $A \subseteq P$ is a set of shareholders, then M_A denotes the restriction of M to rows jointly owned by A. We denote d_A for the number of rows in M_A. Similarly, for $\boldsymbol{s} \in Z^d$ let $\boldsymbol{s}_A \in Z^{d_A}$ denote the restriction of $\boldsymbol{s}$ to the coordinates jointly owned by A. The share of shareholder j

[3] We note that the BBSS constructions of [13, 8] are in fact applicable to distributed exponentiation, but this is due to special properties of those constructions.

is then defined to be $\boldsymbol{s}_{\psi^{-1}(j)}$, which denotes all the entries in $\boldsymbol{s}$ which shareholder j owns, i.e., the share of shareholder j is the share units owned by j.

More formally, we let $[0..2^l]$ be the set of secrets, then each shareholder j is associated a positive integer $d_j = |\psi^{-1}(j)|$ for $1 \leq j \leq n$, such that the set of possible shares for shareholder j, is a subset $\mathcal{S}_j \subseteq Z^{d_j}$ of the Z-module Z^{d_j}. Each possible share for shareholder j is in the subset $\mathcal{S}_j$. The size of shareholder j share is defined to be the number of bits used to uniquely represent the share from $\mathcal{S}_j$. Note, that $d = \sum_{j=1}^{n} d_j$, where d is the number of share units. Then let $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n \subseteq Z^d$, which defines the subset of possible shares for the shareholders. Define the *expansion rate* to be $\mu = d/n$, where d is the number of share units and n is the number of shareholders. Note, that for a given distribution of a secret, the shares of the shareholdes can be considered as an element in the subset $\mathcal{S}$. If we use m bits to uniquely represent the shares in $\mathcal{S}$, then we define the *average share size* to be m/μ, which is the number of share bits each shareholder will get on average.

Definition 3. *A LISS scheme is* correct*, if the secret is reconstructed from shares $\{s_i \mid i \in A\}$ where A is a qualified set of shareholders, by taking an integer linear combination of the shares, with coefficient that depend only on the index set A.*

Definition 4. *A LISS scheme is* private*, if for any two secrets s, s', independent random coins r, r' and any forbidden set A of shareholders, the distribution of $\{s_i(s, r, k) \mid i \in A\}$ and $\{s_i(s', r', k) \mid i \in A\}$ are statistically indisinguishable. More precisely, the statistical distance between the two distributions is negligible in k.*

In the following we define the notion of an *Integer Span Program* (ISP, introduced in [8]) and show how any ISP can be used to build a correct and private LISS scheme.

Definition 5. *$\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ is called an* Integer Span Program *(ISP), if $M \in Z^{d \times e}$ and the d rows of M are labelled by a surjective function $\psi : \{1, \ldots, d\} \to \{1, \ldots, n\}$. Finally, $\boldsymbol{\varepsilon} = (1, 0, \ldots, 0)^T \in Z^e$ is called the* target vector*. We define* $\text{size}(\mathcal{M}) = d$*, where d is the number of rows of M.*

Definition 6. *Let Γ be a monotone access structure and let $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ be a integer span program. Then $\mathcal{M}$ is an ISP for Γ, if for all $A \subseteq \{1, \ldots, n\}$ the following holds.*

- *If $A \in \Gamma$, then there is a vector $\boldsymbol{\lambda} \in Z^d$ such that $M_A^T \boldsymbol{\lambda} = \boldsymbol{\varepsilon}$.*
- *If $A \notin \Gamma$, then there exists $\boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_e)^T \in Z^e$ such that $M_A \boldsymbol{\kappa} = \mathbf{0} \in Z^d$ with $\kappa_1 = 1$, which is called the* sweeping vector *for A.*

In other words, the rows owned by a qualified set must include the target vector in their span, while for a forbidden set, there must exist a sweeping vector which is orthogonal to all rows of the set, but has inner product 1 with the target vector. We also say that $\mathcal{M}$ computes Γ.

We define $\kappa_{\max} = \max\{|a| \mid$ *a is an entry in some sweeping vector*$\}$

Note 1. In the case of a *span program*, which works over a field, the explicit requirement of a sweeping vector is not necessary. This is because the following holds for fiels, $\boldsymbol{\varepsilon} \in \operatorname{im}(M_A^T)$ if and only if there exists a sweeping vector. When working with the integers then only the "only if" implication is guaranteed.

If we have an ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ which computes Γ, we build a LISS scheme for Γ as follows: we use M as the distribution matrix, and set $l_0 = l + \lceil \log_2(\kappa_{\max}(e-1)) \rceil + 1$, where as before l is the length of the secret.

Now, the first requirement in Definition 6 obviously makes the scheme correct, in that a qualified set A can compute the secret by taking a linear combination of their values, since there exsists $\boldsymbol{\lambda}_A \in Z^{d_A}$ such that $M_A^T \cdot \boldsymbol{\lambda}_A = \boldsymbol{\varepsilon}$ which gives

$$\boldsymbol{s}_A^T \cdot \boldsymbol{\lambda}_A = (M_A \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_A = \boldsymbol{\rho}^T \cdot (M_A^T \cdot \boldsymbol{\lambda}_A) = \boldsymbol{\rho}^T \cdot \boldsymbol{\varepsilon} = s$$

The Lemma below shows that the second requirement is sufficient to make the scheme private.

Lemma 1. *If $s \in [0..2^l]$ and the ρ_i's are chosen uniformly at random in $[0..2^{l_0+k}]$ for all $2 \leq i \leq e$, then the LISS scheme derived from $\mathcal{M}$ is private.*

Proof. We have chosen $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$, with $\rho_i \in [0..2^{l_0+k}]$ as uniformly random numbers for $2 \leq i \leq e$, and the secret $s \in [0..2^l]$.

Let $s' \in [0..2^l]$ be arbitrary. We first observe, that $\boldsymbol{s}_A = M_A \boldsymbol{\rho}$ are shares that a subset A can see. If $A \notin \Gamma$, then we by definition know that there exists a sweeping vector $\boldsymbol{\kappa}$ such that $M_A \boldsymbol{\kappa} = \mathbf{0} \in Z^{d_A}$.

Define $\boldsymbol{s}' = M(\boldsymbol{\rho} + (s' - s)\boldsymbol{\kappa})$. We note that $\boldsymbol{s}'_A = \boldsymbol{s}_A$, i.e., the shareholders in A see the same shares, but the secret s' was shared instead of s. Define $\boldsymbol{\rho}$ to be *good* if $\boldsymbol{\rho}' = \boldsymbol{\rho} + (s' - s)\boldsymbol{\kappa}$ has entries in the specified range. Then the above implies that if we restrict the distribution of A's shares of s to the cases where $\boldsymbol{\rho}$ is good, the resulting distribution equals the one generated from s' and $\boldsymbol{\rho}'$.

It follows that the statistical distance between the distributions of A's shares of s and s' is at most twice the probability that $\boldsymbol{\rho}$ is not good, which we can estimate by the union bound as $e - 1$ times the probability that a single entry is out of range. So since $|s' - s| \leq 2^l$, the distance is at most

$$2 \cdot \frac{2^l \kappa_{\max}(e-1)}{2^{l_0+k}} \leq 2^{-k}$$

3 Constructions

3.1 Benaloh-Leichter

In this section we show how to construct an ISP based on Benaloh and Leichter Generalized Secret Sharing scheme [1]. This scheme was already shown to work for secret sharing in any finite group, but to use it over the integers, we need to revisit the scheme to make sure that the required sweeping vectors exist and check the size of their coordinates.

As pointed out in [1], there is a one-to-one correspondence between monotone access structures and monotone formulas. Every monotone access structure can be described by a monotone formula, and every monotone formula describes a monotone access structure, where each variable in the formula is associated with a shareholder in P. A subset of the shareholders corresponds to an input to the formula by setting an input variable to 1 iff the corresponding shareholder is in the the subset. A subset is in the access structure represented by the formula if the formula accepts the corresponding setting of the variables. So it is enough to show how to construct an ISP from an arbitrary monotone formula f.

The details of this follow from Benaloh and Leichter's original construction and can be found in the full version of this paper [14]. Here, we only summarize the conclusions:

One can efficiently construct a distribution matrix $M \in Z^{d\times e}$ for the access structure representing monotone formula f, where d, e are at most the size of f. Moreover, each row has only 0 or 1 entries and there are at most $depth(f)$ 1's in every row. Finally, sweeping vectors have only $0, 1, -1$ as entries.

So when sharing a secret using M we need at most $d \cdot \mathrm{depth}(f)$ additions to calculate all the d share units from (1). Each share unit is the result of adding at most of $\mathrm{depth}(f)$ integers of $(l_0 + k)$-bit, i.e., each share unit is at most $l_0 + k + \log \mathrm{depth}(f)$ bits long.

From [23] we have the existence of a monotone formula for the majority function of size $\mathcal{O}\left(n^{5.3}\right)$ and of depth $\mathcal{O}\left(\log n\right)$. A threshold-$t$ function $T_{t,n}$ can be constructed from the majority function, by fixing some of the inputs of the majority function. This construction implies that we need a majority function of size at most $2n$ to construct the threshold-t function $T_{t,n}$, i.e. [23] gives the existence of a monotone formula for the threshold-t function $T_{t,n}$ of size $\mathcal{O}\left(n^{5.3}\right)$ and of depth $\mathcal{O}\left(\log n\right)$.

It follows from the above that each share unit is of size $\mathcal{O}\left(l_0 + k + \log\log n\right)$ and the time to compute all share units is $\mathcal{O}\left(n^{5.3} \log n (l_0 + k + \log\log n)\right)$, where we assume it takes $\mathcal{O}\left(b\right)$ time to add two b-bit numbers and $\mathcal{O}\left(b\right)$ time to generate a b-bit random integer. This implies that the average share size is $\mathcal{O}\left(n^{4.3}(l_0 + k + \log\log n)\right)$ bits.

Boppana, generalizing Valiant's result in [2], showed that every threshold t function $T_{t,n}$ can be represented by a monotone formula of size $\mathcal{O}\left(t^{4.3} n \log n\right)$. Each share unit size is still the same, hence the average share size becomes $\mathcal{O}\left(t^{4.3} \log n (l_0 + k + \log\log n)\right)$ bits. The total computation time of alle the shares is $\mathcal{O}\left(t^{4.3} n \log^2 n (l_0 + k + \log\log n)\right)$.

3.2 Cramer-Fehr

In this section we consider the ISP's constructed by Cramer and Fehr in [8].

As described, if we have an ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ we use $M \in Z^{d\times e}$ as the distribution matrix and we calculate the shares from (1). If we define $m_{\max}$ to be the maximal entry in the distribution matrix M. We need $d \cdot e$ multiplications of $\mathcal{O}\left(l_0 + k + m_{\max}\right)$-bit numbers and $d\cdot(e-1)$ additions of $\mathcal{O}\left(l_0 + k + m_{\max} + e\right)$-bit numbers to calculate the shares.

From the proof of Corollary 1 in [8] we have that for a threshold-t access structure $T_{t,n}$ that

$$\begin{aligned} d &= n(\lfloor \log n \rfloor + 2) \\ e &= t(\lfloor \log n \rfloor + 2) + 1 \end{aligned}$$

We also know, that $m_{\max} = \mathcal{O}\left(n^2\right)$. If we assume that we use $\mathcal{O}(b)$ time to choose a b-bit random number, $\mathcal{O}(b)$ time to add two b-bit numbers, and $\mathcal{O}\left(b \log^2 b\right)$ time to multiply two b-bit numbers. Then we need

$$\begin{aligned} &\mathcal{O}\left(tn \log^2 n(l_0 + k + n^2)\log^2(l_0 + k + n^2) + tn\log^2 n(l_0 + k + n^2 + t\log n)\right) \\ &= \mathcal{O}\left(tn\log^2 n(l_0 + k + n^2)\log^2(l_0 + k + n^2)\right) \end{aligned}$$

time to compute the shares. Furthermore, we have that each share unit is of size $\mathcal{O}\left(l_0 + k + n^2 + \log(t \log n)\right) = \mathcal{O}\left(l_0 + k + n^2\right)$-bit, hence the average share size is $\mathcal{O}\left(\log n(l_0 + k + n^2)\right)$.

3.3 Comparison

In this section we compare the average share size, the number of random bits required to do the computations, and the computation complexity of the LISS scheme based on Benaloh-Leichter construction (BLc) with the scheme based on the Cramer-Fehr construction (CFc) in the threshold-t case.

First we make some observations. Recall that $l_0 = l + \lceil \log_2(\kappa_{\max}(e-1)) \rceil + 1$. For BLc we get that $l_0 = l + \lceil \log_2(n^{5.3} - 1) \rceil + 1$, which asymptotically reduces to $l_0 \in \mathcal{O}(l + \log n)$. In the CFc we have that $\kappa_{\max} = c2^n$ and $e = t(\lfloor \log n \rfloor + 2) + 1$, i.e., $l_0 = l\lceil \log_2(c2^n t(\lfloor \log n \rfloor + 2)) \rceil + 1$, which asymptotically reduces to $l_0 \in \mathcal{O}(l + n)$.

First we will compare the results of the CFc with the BLc based on the threshold-t function build from Valiant [23] majority function. The results are compared in the table below, where we use l instead of the more scheme dependent l_0. Let *ss* denote the share size of each shareholder, *rb* the number of random bits used in the computation of the shares, and *ct* the computation time of the shares.

	CFc	BLc (Valiant)
ss	$\mathcal{O}\left((l + k + n^2)\log n\right)$	$\mathcal{O}\left((l + k + \log\log n)n^{4.3}\right)$
rb	$\mathcal{O}\left((l + k + n)t\log n\right)$	$\mathcal{O}\left((l + k + \log\log n)n^{5.3}\right)$
ct	$\mathcal{O}\left(tn\log^2 n(l + k + n^2)\log^2(l + k + n^2)\right)$	$\mathcal{O}\left(n^{5.3}\log n(l + k + \log\log n)\right)$

These results show a great advantage of the CFc if n is a dominating factor of the parameters, if this is not the case, the asymptotic bounds are of the same magnitute.

We may also base the BLc on the result from Boppana [2], which states that the size of the formula for the threshold function $T_{t,n}$ is $\mathcal{O}\left(t^{4.3} n \log n\right)$. We now compare it against the CFc and let t be fixed while n grows. This implies that the formula size is $\mathcal{O}(n \log n)$ for a fixed value of t. This can be a reasonable model in some cases: we may have a large number of share holders, while we believe that the adversary can only corrupt a small number of them. In the table below we compare the results to the CFc for a constant value of t,

	CFc	BLc (Boppana)
ss	$\mathcal{O}\left((l+k+n^2)\log n\right)$	$\mathcal{O}\left((l+k+\log\log n)\log n\right)$
rb	$\mathcal{O}\left((l+k+n)\log n\right)$	$\mathcal{O}\left((l+k+\log\log n)n\log n\right)$
ct	$\mathcal{O}\left(n\log^2 n(l+k+n^2)\log^2(l+k+n^2)\right)$	$\mathcal{O}\left(n\log^2 n(l+k+\log\log n)\right)$

Note that in this case, the BLc actually has a better share size and computation time complexity than the CFc. This indicates that the BLc with the current state of the art can compete with the CFc in special cases.

Results of Radhakrishnan [3] show that the lower bound for a monotone formula that computes the threshold-t function $T_{t,n}$ for $2 \leq t \leq \frac{n}{2}$, has size at least $\lfloor \frac{t}{2} \rfloor n \log(\frac{n}{t-1})$. As he notes, that in the monotone formulas model, the complexities of computing $T_{t,n}$ and $T_{n-t+1,n}$ are the same. Hence, the lower bound of $\lfloor \frac{t}{2} \rfloor n \log(\frac{n}{t-1})$ holds for the function $T_{n-t+1,n}$, $2 \leq t \leq \frac{n}{2}$, as well. This result is far below Valiants [23] and Boppana [2], so in particular BLc is in general better than CFc if the bound turns out to be tight.

To summarize the results of this section, we find that CFc seems better in the general case of the threshold-t function, but if n is small compared to the other factors, then the BLc can be just as good. Furthermore, for fixed t and large n, the BLc has an advantage over the CFc. The result of Radhakrishnan gives a big gab for improvements from the current state of the art of threshold functions, which would favor BLc. Finally, it must be stressed, of course, that the BLc has the advantage that it can be used over any monotone access structure. However, the BLc is only efficient if there is a polynomial-size monotone formula describing the access structure.

4 Distributed Exponentiation

In this section we will consider solutions to the the distributed exponentiation problem based on LISS. The set-up is as follows: we have n servers $P_1, ..., P_n$, an access structure Γ with an ISP $\mathcal{M} = (M, \psi, \varepsilon)$, and an adversary *Adv* who may corrupt any subset of servers not in Γ. The family of subsets not in Γ is called *the adversary structure* $\bar{\Gamma}$. Finally, we have a special player C called the client who may also be corrupted, independently of which servers are corrupt

In this first solution we give, we consider non-adaptive corruption in the semihonest model, i.e., the adversary must choose which players to corrupt before the protocol starts, he sees all internal data and communication of corrupt players, he may cause them to stop playing at any time, but all players follow the protocol as long as they participate. In order to solve the problem in this model, we must assume that the adversary structure is Q2, i.e., any set of form $A \cup B$, $A, B \in \bar{\Gamma}$ is strictly smaller than $\{P_1, ..., P_n\}$. This ensures that the set of honest servers is in Γ.

We will use Canetti's Universal Composability (UC) framework to state and prove our protocols. For details on this framework, refer to [4]. In order to focus on the actual protocol for exponentiation, we will assume a trusted dealer who chooses the group to use and secret-shares the exponent. In the UC framework, this means we assume a functionality representing the dealer is given, as detailed below. We assume for simplicity synchronous communication and also that the client C can broadcast information to all servers. But we do not assume any private channels so all communication between players is seen by the adversary.

Functionality F_{Deal}

1. Upon receiving "start" from all honest players, choose the group G to use and an exponent s (in principle any efficient algorithm for this could be used here).
2. Generate the distribution vector $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$ and calculate the shares from
$$M \cdot \boldsymbol{\rho} = (s_1, \ldots, s_d)^T,$$
finally distributes the shares, such that s_i is sent privately to $P_{\psi(i)}$ for $1 \leq i \leq d$. Finally, send a description of G to all players and the adversary (information allowing to represent group elements and computing the group operation).

Such a functionality together with the protocol we give below will implement the following functionality

Functionality $F_{Deal-and-Exp}$

1. Upon receiving "start" from all honest players, choose the group G to use and an exponent s (same algorithm as used in F_{Deal}). Send a description of G (information allowing to represent group elements and computing the group operation) to all players and the adversary.
2. At any later time, upon receiving "Exponentiate a" for $a \in G$ from the client, send "Exponentiate a", to all players and the adversary. In the next round, send "Result a^s" to the client and the adversary.

The protocol proceeds as follows:

Protocol Exponentiate

1. Initially, each player sends "start" to F_{Deal}, and stores the description of G and shares of s received from F_{Deal}.
2. On input $a \in G$, C broadcasts a to the servers.
3. Each P_j sends to C $a_i = a^{s_i}$ for each component s_i of the share held by P_j.
4. Since $\bar{\Gamma}$ is Q2, C is guaranteed to receive valid contributions from a qualified set of players $A \in \Gamma$. C uses the entries in the reconstruction vector for A $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{d_A})^T$ together with the contributions $(a_1 = a^{s_1}, \ldots, a_{d_A} = a^{s_{d_A}})$ to construct
$$a^s = \Pi_{i=1}^{d_A} a_i^{\lambda_i}.$$

Theorem 1. *The Exponentiate protocol when given access to F_{Deal} and a broadcast channel from C to the servers, securely implements $F_{Deal-and-Exp}$. The adversary is assumed to non-adaptively corrupt any set in Q2 structure $\bar{\Gamma}$ in a semi-honest fashion.*

Proof. Security is proved by constructing an ideal model adversary which works in a setting where it may communicate with ideal functionality $F_{Deal-and-Exp}$ and must simulate everything the real life adversary Adv would see in a real attack. This works by running internally a copy of Adv and proceeds as follows:

1. Let B be the set of servers corrupted by Adv. Having received the description of G from $F_{Deal-and-Exp}$, compute a sharing of 0 to simulate the action of F_{Deal}, i.e., the distribution vector is $\boldsymbol{\rho} = (0, \rho_2, \ldots, \rho_e)^T$ and the shares are
$$\boldsymbol{s} = (s_1, \ldots, s_d)^T = M \cdot \boldsymbol{\rho} \qquad (2)$$
Give to the Adv the shares from (2) belonging to the servers in B.
2. Upon receiving "Exponentiate a" and "Result a^s" from $F_{Deal-and-Exp}$, we must simulate the contributions that honest players send to C. To this end, note that if we had used $\boldsymbol{\rho}' = \boldsymbol{\rho} + s\boldsymbol{\kappa}_B$ as distribution vector in (2), then the corrupted servers in B would get the same shares, but the secret value would be s instead of 0.
Now, let R be a row in the distribution matrix M belonging to honest server P_j, say the i'th row, and let s_i be the share unit we computed from this row in (2). Had we used $\boldsymbol{\rho}'$ instead of $\boldsymbol{\rho}$, then the share unit coming from R would have been $s_i' = (\boldsymbol{\rho} + s\boldsymbol{\kappa}_B) \cdot R = s_i + s\boldsymbol{\kappa}_B \cdot R$ instead. The observation is now that because we know a^s and s_i, we can compute $a^{s_i'}$ even though we do not know s. Concretely, we simulate the contribution from P_j by
$$\begin{aligned} a^{s_i}(a^s)^{\boldsymbol{\kappa}_B \cdot R} &= a^{s_i + s\boldsymbol{\kappa}_B \cdot R} \\ &= a^{s_i'} \end{aligned}$$
Give all simulated contributions to Adv.

We now need to prove that no environment can distinguish between the real protocol and the simulated game. The is straightforward: First, the shares computed in step 1 of the simulation are statistically indistinguishable from the shares computed by F_{Deal} by privacy of the LISS scheme and since B in unqualified. Second, in both the simulated game and real protocol, honest players output always the correct value a^s, by definition of $F_{Deal-and-Exp}$, respectively correctness of the LISS scheme. Finally, given a, a^s, the simulated and real contributions from honest players are statistically indistinguishable, since the vector we use for the simulated sharing is $\boldsymbol{\rho}' = \boldsymbol{\rho} + s\boldsymbol{\kappa}_B$ which is statistically close to a uniformly chosen sharing vector for s. □

4.1 Active Adversaries and Distributed RSA

If we are not guaranteed that corrupted players follow the protocol, we can expand the Exponentiate protocol in a natural way by having players prove in zero-knowledge that their contributions are correct. Given any appropriate scheme for proving correctness of contributions, a corrupt player must either give correct information or be disqualified. Since this is equivalent to the semihonest model, security essentially follows from security of the zero-knowledge proofs and the proof we already gave above.

Depending on the structure of the group and the assumptions we are willing to make, there are many different ways to do the zero-knowledge proofs, see for instance [21, 19, 22, 7, 11, 10, 12, 16]. Most of the techniques can be made non-interactive in the random oracle model, or are already non-interactive given some set-up assumption. If all else fails, generic zero-knowledge techniques can be used[17].

However, a detailed account of all possibilities is out of scope of this paper. We concentrate instead on distributed RSA as a particularly interesting special case. The functionality for initial set-up and the functionality we want to implement are modified from the general case as follows:

Functionality $F_{RSA-Deal}$

1. Upon receiving "start" from all honest players, choose the modulus n to use, secret and public exponents s, e and a random square v in $G = Z_N^*$.
2. Generate the distribution vector $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$ and calculate the shares from
$$M \cdot \boldsymbol{\rho} = (s_1, \ldots, s_d)^T,$$
finally distributes the shares, such that s_i is sent privately to $P_{\psi(i)}$ for $1 \leq i \leq d$. Finally, send N, e, v and $v_i = v^{s_i} \bmod N$ for every share unit s_i to all players and the adversary .

Functionality F_{RSA}

1. Upon receiving "start" from all honest players, choose the modulus N to use, secret and public exponents s, e. Send N, e to all players and the adversary
2. At any later time, upon receiving "Exponentiate a" for $a \in Z_N^*$ from the client, send "Exponentiate a", to all players and "Exponentiate $a, a^s \bmod N$" to the adversary. Two rounds later, send "Result $a^s \bmod N$" to all players.

The protocol we will use is the Exponentiate protocol from the previous section, with the extension that C will check each contribution $a_i = a^{s_i} \bmod N$ from server P_j. We want to show that a sufficient check can be done in constant-round without using random oracles to ensure soundness and zero-knowledge, and regardless of which modulus and public exponent is used. To do this, we generalize the results from [10]. Concretely, we use the following well known protocol, which we will repeat in parallel $\lceil 2 + 2 \log_2 n \rceil$ times:

1. P_j chooses a random $k + max$- bit number r and sends to C $u_1 = a^r \bmod N, u_2 = v^r \bmod N$. Here, max is the maximal bitlength of any s_i that can occur.
2. C sends a random bit b to P_j.

3. P_j sends $z = r + bs_i$, and C checks that $a^z = u_1 a_i^b \bmod N, v^z = u_2 v_i^b \bmod N$

The following Lemma is an easy consequence of corresponding results in [10]:

Lemma 2. *The above protocol is statistical zero-knowledge. Furthermore, if $a_i \neq a^s \bmod N$ then a polynomial time prover who can make C accept with probability more than $1/(4n^2)$ can compute efficiently a multiple of the order of v.*

Note that the last result in the lemma implies that if an adversary can cheat the protocol on input a random v, he can factor N by a standard reduction and hence also break RSA.

Even though the soundness error for this protocol is not negligible, we can show that checking the contributions in this way is sufficient to allow C to reconstruct the correct result efficiently. This is done by a generalization of the results from [10]. There it was observed that as long as the expected number of accepted incorrect contributions is small enough, C can reconstruct efficiently by searching exhaustively for a set of correct contribution. In [10], this was done for the case of a threshold access structure. Here we have to be more careful with the search algorithm and the analysis because we have no lower bound on the number of honest players for a general access structure.

Algorithm Reconstruct

1. On input public key $N, e, a \in Z_N^*$ and a set of contributions to finding $a^s \bmod N$, execute the protocol above with each server to check the correctness of each contribution.
2. Let the set of accepted contributions be *Acc*. Do the following for $j = 0, ..., |Acc|$:
3. For each subset $B \subset Acc$ of size $|Acc| - j$, run the reconstruction algorithm from the Exponentiate protocol on the contributions in B, attempting to compute $a^s \bmod N$. Let z be the result. If $z^e = a \bmod N$, output z and stop.

Lemma 3. *The expected number of subsets considered by Reconstruct is at most 2.*

Proof. Let m be the number of incorrect contributions submitted by corrupt players. Clearly, the worst case is if all corrupt players submit bad contributions, so we may assume that the number of honest players is $n - m$. Let p be the probability that an incorrect contribution is accepted. Then

$$p_i = Pr(i \text{ incorrect shares accepted}) = p^i(1-p)^i \binom{m}{i} \leq p^i m^i$$

Given that i incorrect shares are accepted, we have $n - m + i$ contributions, and we finish at the latest when we have searched all subsets of size $n - m$. This means checking a total of

$$\begin{aligned}\binom{n-m+i}{n-m+i} + \binom{n-m+i}{n-m+i-1} + \dots + \binom{n-m+i}{n-m} &\leq (i+1)\binom{n-m+i}{n-m} \\ &= (i+1)\binom{n-m+i}{i} \\ &\leq (i+1)(n-m+i)^i\end{aligned}$$

subsets. It follows that the expected number of subsets we check is at most

$$\sum_{i=0}^{m} p^i m^i \cdot (i+1)(n-m+i)^i \leq \sum_{i=0}^{m} p^i m^i 2^i n^i \leq \sum_{i=0}^{m} (2pn^2)^i \leq \sum_{i=0}^{m} 2^{-i} \leq 2$$

using the above and the fact that $p \leq 1/(4n^2)$. □

A final observation is that by choosing z at random in Z_N^*, and setting $v = z^{2e} \bmod N$, a simulator can easily create a random square v for which $v^s \bmod N$ is known (namely $z^2 \bmod N$). It is then easy to simulate the information $F_{RSA-Deal}$ sends to corrupt players. Using this, the proof of Theorem 1, Lemma 3 and Lemma 2, it is straightforward to show:

Theorem 2. *Under the RSA assumption, the Exponentiate protocol expanded with the above Reconstruction algorithm and given access to the $F_{RSA-deal}$ functionality implements the F_{RSA} functionality. The adversary may non-adaptively and actively corrupt any set in Q2 structure $\bar{\Gamma}$.*

We believe that the interest of this result is that it buys us full generality in access structure and choice of keys and no dependency on extra set-up or complexity assumptions. Since the number of servers n can be expected to be quite small in practice, the overhead compared to more standard solutions is moderate: a factor of $\log n$ in complexity and potentially 2 extra moves. However, in practice, faults are usually rare, so if the the client attempts to get the result from all contributions first and only asks to have the proofs completed if this fails, then the scheme will be non-interactive "almost always".

Acknowledgements

Thomas Mølhave, Peter Bro Miltersen, Gudmund Skovbjerg Frandsen, and the anonymous reviewers from the committee for helpful comments.

References

1. Josh Cohen Benaloh, Jerry Leichter: *Generalized Secret Sharing and Monotone Functions.* Proc. of CRYPTO 1988: 27-35
2. R. B. Boppana: *Amplification of Probabilistic Boolean Formulas.* Advances in Computing Research 5: 27-45 (1989)

3. Jaikumar Radhakrishnan: *Better Lower Bounds for Monotone Threshold Formulas.* J. Comput. Syst. Sci. 54(2): 221-226 (1997)
4. Ran Canetti: *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, FOCS 2001: 136-145.
5. Benny Chor and Eyal Kushilevitz: *Secret Sharing Over Infinite Domains.*, J. Cryptology 6(2): 87-95 (1993)
6. Benny Chor, Mihály Geréb-Graus and Eyal Kushilevitz *Private Computations over the Integers.*, SIAM J. Comput. 24(2): 376-386 (1995)
7. Cramer and Damgård: *Secret-Key Zero-Knowlegde and Non-interactive Verifiable Exponentiation*, Proc. of TCC 04, Springer Verlag LNCS.
8. Ronald Cramer, Serge Fehr: *Optimal Black-Box Secret Sharing over Arbitrary Abelian Groups.* Proc. of CRYPTO 2002: 272-287
9. Cramer, Fehr and Stam: *Black-Box Secret Sharing from Primitve Sets in Algebraic Number Fields*, Proc. of Crypto 05, Springer Verlag LNCS.
10. Ivan Damgård, Kasper Dupont: *Efficient Threshold RSA Signatures with General Moduli and No Extra Assumptions.* Proc. of Public Key Cryptography 2005: 346-361
11. Damgård, Fazio and Nicolosi: *Non-Interactive Zero-Knowledge Proofs from Homomorphic Encryption* Proc. of TCC 06, Springer Verlag LNCS.
12. Ivan Damgård, Maciej Koprowski: *Practical Threshold RSA Signatures without a Trusted Dealer.* Proc. of EUROCRYPT 2001: 152-165
13. Yvo Desmedt, Yair Frankel: *Perfect Homomorphic Zero-Knowledge Threshold Schemes over any Finite Abelian Group.* SIAM J. Discrete Math. 7(4): 667-679 (1994)
14. Ivan Damgård and Rune Thorbek: *Linear Integer Secret Sharing and Distributed Exponentiation* (full version), the Eprint archive, www.iacr.org
15. Yair Frankel, Peter Gemmell, Philip D. MacKenzie, Moti Yung: *Optimal Resilience Proactive Public-Key Cryptosystems.* FOCS 1997: 384-393
16. Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk: *Robust and Efficient Sharing of RSA Functions.* J. Cryptology 2000 13(2): 273-300
17. Oded Goldreich, Silvio Micali, Avi Wigderson: *Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems* J. ACM 38(3): 691-729 (1991).
18. Tal Rabin: *A Simplified Approach to Threshold and Proactive RSA.* Proc. of CRYPTO 1998: 89-104
19. Claus-Peter Schnorr: *Efficient Signature Generation by Smart Cards.* J. Cryptology 4(3): 161-174 (1991)
20. Adi Shamir: *How to Share a Secret.* Commun. ACM 22(11): 612-613 (1979)
21. Alfredo De Santis, Yvo Desmedt, Yair Frankel, Moti Yung: *How to share a function securely. STOC 1994: 522-533*
22. Victor Shoup: *Practical Threshold Signatures.* Proc. of EUROCRYPT 2000: 207-220
23. Leslie G. Valiant: *Short Monotone Formulae for the Majority Function.* J. Algorithms 5(3): 363-366 (1984)