Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 23 (2016)

How to Share a Secret, Infinitely

Ilan Komargodski * Moni Naor * Eylon Yogev *

Abstract

Secret sharing schemes allow a dealer to distribute a secret piece of information among several parties so that any qualified subset of parties can reconstruct the secret, while every unqualified subset of parties learns nothing about the secret. The collection of qualified subsets is called an access structure. The best known access structure is the k-threshold one, where any subset of size k or more is qualified and all smaller subsets are not qualified. A major issue is the size of the shares needed to assure this property. For the threshold access structure, when $k = 2$ and there are n parties it is known that the share size must be $\log n$ even for secrets of 1 bit.

In this work we consider the case where the set of parties is not known in advanced and could potentially be infinite. We would still like to give the t^{th} party arriving a small share as possible as a function of t. We show that for any access structure it is possible to do so by giving shares of size 2^{t-1}. Our main result is a scheme for k-threshold access structure with shares of size $(k-1) \cdot \log t + \mathsf{poly}(k) \cdot o(\log t)$ bits. Finally, we prove that no secret sharing scheme for the 2-threshold access structure with share size at most $\log t + \log \log t + O(1)$ exists.

*Weizmann Institute of Science, Israel. Email: {ilan.komargodski,moni.naor,eylon.yogev}@weizmann.ac.il. Supported in part by a grant from the I-CORE Program of the Planning and Budgeting Committee, the Israel Science Foundation, BSF and the Israeli Ministry of Science and Technology. Moni Naor is the incumbent of the Judith Kleeman Professorial Chair. Ilan Komargodski is supported in part by a Levzion fellowship.

ISSN 1433-8092

1 Introduction

640K ought to be enough for anybody

Misattributed to Bill Gates, 1981

Engineering scalable systems is a delicate business: important decisions have to be made regarding balancing scalability and efficiency when fixing system parameters (such as the representation size of a date, the number of clients the system can serve simultaneously, security parameters and more). This inherent tradeoff between scalability and efficiency has had devastating consequences. There are many Y2K [Wikb] style horror stories such as losing contact with the NASA space probe "Deep Impact" when its internal clock reached exactly 2^{32} one-tenth seconds since 2000 on 11 August 2013, 00:38:49 [Geo] and the IPv4 address exhaustion problems caused by the limited allocation size for numeric Internet addresses [Wika]. Can we design scalable systems without suffering a great deal of efficiency costs? In this work we investigate methods that do not assume a fixed upper bound on the number of participants in the area of secret sharing.

Secret sharing is a method by which a secret piece of information can be distributed among n parties so that any qualified subset of parties can reconstruct the secret, while every unqualified subset of parties learns nothing about the secret. The collection of qualified subsets is called an access structure. Secret sharing schemes are a basic primitive and have found applications in cryptography and distributed computing; see the extensive survey of Beimel [Bei11]. A significant goal in secret sharing is to minimize the share size, namely, the amount of information distributed to the parties.

Secret sharing schemes were introduced in the late 1970s by Shamir [Sha79] and Blakley [Bla79] for the k-out-of-n threshold access structures that includes all subsets of cardinality at least k for $1 \leq k \leq n$. Their constructions are fairly efficient both in the size of the shares and in the computation required for sharing and reconstruction. Ito, Saito and Nishizeki [ISN93] showed the existence of a secret sharing scheme for every (monotone) access structure. In their scheme the size of the shares is proportional to the depth 2 complexity of the access structure when viewed as a Boolean function (and hence shares are exponential for most structures). Benaloh and Leichter [BL88] gave a scheme with share size polynomial in the monotone *formula* complexity of the access structure. Karchmer and Wigderson [KW93] generalized this construction so that the size is polynomial in the monotone span program complexity.

All of these schemes require that an upper bound on the number of participants is known in advance. However, in many scenarios this is either unrealistic or prone to disaster. Moreover, even if a crude upper bound n is known in advance, it is preferable to have as small shares as possible if the eventual number of participants is much smaller than n.

In this work we consider the well motivated, yet almost unexplored[1], case where the set of parties is *not* known in advanced and could potentially be infinite. Our goal is to give the t^{th} party arriving a share small as possible as a function of t. We require that in each round, as a new party arrives, there is no communication to the parties that have already received shares, i.e., the dealer distributes a share only to the new party. We call such access structures evolving: the parties arrive one by one and, in the most general case, a qualified subset is revealed to the dealer only when all parties in that subset are present (in special cases the dealer knows the access structure to begin with, just does not have an upper bound on the number of parties). For this to make sense we

[1] But see the work of Csirmaz and Tardos [CT12] discussed below.

assume that the changes to the access structure are monotone, namely, parties are only added and qualified sets remain qualified.

Our first result is a construction of a secret sharing scheme for *any* evolving access structure.

Theorem 1.1. *For every evolving access structure there is a secret sharing scheme where the share size of the t^{th} party is 2^{t-1}.*

Then, we construct more efficient schemes for specific access structures. We focus on the evolving k-threshold access structure for $k \in \mathbb{N}$, where at any point in time any k parties can reconstruct the secret but no $k-1$ parties can learn anything about the secret.

Theorem 1.2 (Informal)**.** *There is a secret sharing scheme for the evolving k-threshold access structure in which the share size of the t^{th} party is $(k-1) \cdot \log t + \mathsf{poly}(k) \cdot o(\log t)$.*

Finally, for $k = 2$, we present a matching lower bound showing that our scheme is tight with respect to the high order term.

Theorem 1.3. *For any constant $c \in \mathbb{N}$, there is* no *secret sharing scheme for the evolving* 2-THR *access structure in which the share size of the t^{th} party is at most $\log t + \log\log t + c$.*

In addition, we present a construction for the evolving 2-threshold access structure with slightly better low order terms. In this scheme the share size of the t^{th} party is $\log t + \log\log t + 2\log\log\log t + 6$. This scheme matches the lower bound even for the second order term.

1.1 Discussion

Schemes for general access structures. In the classical setting of secret sharing many schemes are known for general access structures, depending on their representation [ISN93, BL88, KW93]. All of these schemes result with shares of exponential size for general access structures. One of the most important open problems in the area of secret sharing is to prove the necessity of long shares, namely, find an access structure (even a non-explicit one) that requires exponential size shares.

Our scheme for general evolving access structures also results with exponential size shares. Since any access structure can be made evolving, we cannot hope to obtain anything better than exponential in general (unless we have a major breakthrough in the classical setting).

Threshold schemes. In the classical setting there are several different schemes for the threshold access structure. One of the best such schemes (in terms of the computation needed for sharing and reconstruction and in terms of the share size) is due to Shamir [Sha79]. In this scheme, to share a 1-bit secret among n parties, roughly $\log n$ bits have to be distributed to each party. It is known that $\log n$ bits are essentially required, so Shamir's scheme is optimal (see [CCX13] for the proof and a discussion of the history).

Let us review Shamir's scheme for the k-out-of-n threshold access structure. The dealer holding a secret bit s, samples a random polynomial $p(\cdot)$ of degree $k-1$ with coefficients over $\mathsf{GF}(q)$, where the free coefficient is fixed to be s, and gives party $i \in [n]$ the field element $p(i)$. q is chosen to be the smallest prime (or a power of a prime) larger than n. Correctness of the scheme follows by the fact that k points on a polynomial of degree $k-1$ completely define the polynomial and allow for computing $p(0) = s$. Security follows by a counting argument showing that given less than k points, both possibilities for the free coefficient are equally likely. The share of each party is an

element in the field $\mathsf{GF}(q)$ that can be represented using $\log q \approx \log n$ bits. Notice that the share size is independent of k.

As a first attempt one might try to adapt this procedure to the *evolving* setting. But since n is not fixed, what q should we choose? A natural idea is to use an extension field. Roughly, we would simulate the dealer for Shamir's scheme, sample a random polynomial of degree $k-1$ and increase the field size from which we compute shares as more parties arrive. Ideally, for the share of the t^{th} party we will use a field of size $O(t)$. This implies that the share size of party t would be $\log(O(t)) \ll \log t + \log\log t$ for large enough t. The lower bound in Theorem 1.3 means that *no such solution can work*!

We take a different path for obtaining efficient schemes. For example, for $k = 2$, our scheme results with essentially optimal share size for the t^{th} party: the first two high order terms are $\log t + \log\log t$ (without hidden constant factors) and there is an additional lower order term of $2\log\log\log t + 6$. See the simplified scheme in Section 5.

1.2 Related work

Most similar to our setting is the notion of *on-line* secret sharing of Csirmaz and Tardos [CT12]. Csirmaz and Tardos present a scheme for any access structure in which every party participates in at most d qualified sets, where d is an upper bound known in advance. The share size of every party in this scheme is linear in d. In addition, Csirmaz and Tardos presented a scheme for evolving 2-threshold in which the share size of party t is linear in t. Our Theorem 1.2 is an exponential improvement on the latter.

There are numerous areas where systems are designed to work without any fixed upper bound on the size or the duration they will be used. A few examples include prefix-free encodings of the integers (e.g. Elias codes [Eli75]), labeling nodes in possibility infinite graphs [KNR92] and data structures for approximate set membership for sets of unknown size [PSW13].

1.3 Overview of our constructions and techniques

First, we overview our construction for general evolving access structures. Then, we describe our construction for the evolving 2-threshold access structure. This serves as a warm-up for our more general construction for k-threshold access structures.

Any evolving access structures. Let $\mathcal{A} = \mathcal{A}_1, \mathcal{A}_2, \ldots$ be any evolving access structure with corresponding monotone characteristic functions $f_1, f_2 \ldots$, where $f_t \colon \{0,1\}^t \to \{0,1\}$. Note that the dealer does not know $f_1, f_2 \ldots$ in advance but is only given f_t when the t^{th} party arrives. The main idea underlying our construction is that any monotone function $f_t \colon \{0,1\}^t \to \{0,1\}$ can be written as

$$f_t(x_1, \ldots, x_t) = (x_t \wedge f_t(x_1, \ldots, x_{t-1}, 1)) \vee f_t(x_1, \ldots, x_{t-1}, 0).$$

The important property of this representation is that $f(x_1, \ldots, x_{t-1}, 1)$ and $f(x_1, \ldots, x_{t-1}, 0)$ are independent of x_t, or in other words, the formula that computes $f_{t-1}(x_1, \ldots, x_{t-1})$ is independent of x_t (and all other variables that will appear in the future). This results with a formula for the access structure with the property that the location of a variable depends only on the previous variables (and not of future ones).

To share a secret given this representation of the access structure, we use ideas underlying the scheme of Benaloh and Leichter [BL88] combined with some new ideas. Specifically, our representation differentiates between leaves which are labeled with variables and leaves which are labeled with constants. In the standard setting, where the scheme of Benaloh and Leichter is used, it can be assumed that a formula does not have leaves labeled with constants as they can always be simplified.

The share size of party t in our scheme is equal to the number of times the variable x_t appears as a leaf in the formula. Since the variable x_t appears in our formula 2^{t-1} times, we get a scheme with this share size. See Section 4 for the full details.

Evolving 2-threshold access structure. The approach of [CT12] for the evolving 2-threshold access structure is to give party t a random bit b_t and all bits $s \oplus b_1, \ldots, s \oplus b_{t-1}$. This clearly allows for each pair of parties to reconstruct the secret and ensures that for every single party the secret remains hidden. The share size of the t^{th} party in this scheme is t. (Essentially the same scheme also follows from our general construction in Section 4 with a simple efficiency improvement described towards the end of that section.) Generalizing this idea to larger values of k results with shares of size roughly t^{k-1}.

Whereas the above approach is somewhat naive (and very inefficient in terms of share size), our construction is more subtle and results with exponentially shorter shares. Our main building block is a *domain reduction* technique which allows us to start with a naive solution and apply it only on a small number of parties to get an overall improved construction. Details follows.

We assign each party a generation, where the g^{th} generation consists of 2^g parties (i.e., the generations are of geometrically increasing size). Within each generation we execute a standard secret sharing scheme for 2-threshold. Notice that here we know exactly how many parties are in the same generation: party t is part of generation $g = \lfloor \log t \rfloor$ and the size of that generation is $\text{SIZE}(g) \leq t$. A standard secret sharing scheme for 2-out-of-t costs roughly $\log t$ bits (see Claim 2.3). This solves the case in which both parties come from the same generation.

To handle the case where the two parties come from different generations we use a (possibly naive) scheme for the *evolving* 2-threshold access structure. For each generation we generate *one* share for the evolving scheme and give it to each party in that generation. Thus, if two parties from different generations come together they hold two different shares for the evolving scheme that allow them to reconstruct the secret. Since we generate one share of the evolving scheme per generation, party t holds the share of the $(g = \log t)^{\text{th}}$ party of the evolving scheme!

Summing up, if we start with a scheme in which the share size of the t^{th} party is $\sigma(t)$, then we end up with a scheme with share size roughly $\sigma'(t) = \log t + \sigma(\log t)$. To get our result we start with a scheme in which $\sigma(t) = t$ (described above) and iteratively apply this argument to get better and better schemes.

Evolving k-threshold access structure. There are several ideas underlying the generalization of the 2-threshold scheme to work for any k. As before, we assign each party a generation, but now the g^{th} generation is roughly of size $2^{(k-1)\cdot g}$. This means that party t is in generation $g = \lfloor (\log t)/(k-1) \rfloor$ that includes $\text{SIZE}(g) = t \cdot 2^{k-1}$ parties. Again, within a generation we apply a standard k-out-of-$\text{SIZE}(g)$ secret sharing scheme. This costs us $\log(\text{SIZE}(g)) \leq \log t + k$ bits. This solves the problem if k parties come from the same generation.

We are left with the case where the k parties come from at least two different generations. For this we use a (possibly naive) scheme for the *evolving* k-threshold access structure. For each generation we generate $k-1$ shares $s_1, \ldots, s_{k-1}$ for the evolving scheme and share each s_i using a

standard i-out-of-SIZE(g) secret sharing scheme. Thus, if $\ell \leq k-1$ parties from some generation come together, they can reconstruct $s_1, \ldots, s_\ell$ which are ℓ shares for the evolving scheme. Therefore, any k parties (that come from at least two generations) can reconstruct k shares for the evolving k-threshold scheme that enable them to reconstruct the secret. Since we generate $k-1$ shares of the evolving scheme per generation, party t holds (roughly) the share of the $(\log t + k)^{\text{th}}$ party of the evolving scheme.

The share size needed to share each s_i is $\max\{\log(\text{SIZE}(g)), |s_i|\} \leq \max\{\log t + k, \sigma(\log t + k)\}$. Summing up, if we start with a scheme in which the share size of the t^{th} party is $\sigma(t)$, then we end up with a scheme with share size roughly $\sigma'(t) = \log t + (k-1) \cdot \max\{\log t + k, \sigma(\log t + k)\}$. A small optimization is that sharing s_1 costs just $|s_1|$, as we can give s_1 to each party (similarly to what we did in the $k = 2$ case).

We want to iteratively apply this domain reduction procedure. For this we have to specify the initial scheme. If we start with the scheme that results from the construction in Theorem 1.1 which has share size roughly 2^t (or roughly t^k with the optimization described above), then the resulting scheme will have a factor that depends *exponentially* on k. This makes the scheme impractical even for small values of k.

A formula for the future. To get around this we present a tailor-made construction for the evolving k-threshold in which the share size of party t has *almost linear* dependence on t and k. Specifically, the share size in this scheme is $kt \cdot \log(kt)$. For this, we construct, at least intuitively, a Boolean monotone formula for k-threshold that counts to k.[2] For this counting to make sense in the evolving setting we notice that counting to k can be done by summing up the number of 1's so far with the number of 1's that will come in the future. Since we are counting to k, both of these numbers can be bounded by k, so we have to prepare only k possibilities for the unknown future. To make this construction efficient, we combine it with a generation-like mechanism. See Section 6 for the full details.

2 Model and Definitions

For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. We denote by log the base 2 logarithm and assume that $\log 0 = 0$.

We start by briefly recalling the standard setting of (perfect) secret sharing. Let $\mathcal{P}_n = \{1, \ldots, n\}$ be a set of n parties. A collection of subsets $\mathcal{A} \subseteq 2^{\mathcal{P}_n}$ is monotone if for every $B \in \mathcal{A}$, and $B \subseteq C$ it holds that $C \in \mathcal{A}$.

Definition 2.1 (Access structure)**.** An access structure $\mathcal{A} \subseteq 2^{\mathcal{P}_n}$ is a monotone collection of subsets. Subsets in $\mathcal{A}_n$ are called qualified and subsets not in $\mathcal{A}$ are called unqualified.

Definition 2.2 (Threshold access structure)**.** For every $n \in \mathbb{N}$ and $1 \leq k \leq n$, let (k, n)-THR be the threshold access structure over n parties which contains all subsets of size at least k.

A (standard) secret sharing scheme involves a dealer who has a secret, a set of n parties, and an access structure $\mathcal{A}$. A secret sharing scheme for $\mathcal{A}$ is a method by which the dealer distributes

[2] Even though we can make our construction a monotone formula, our final construction is not phrased as a formula since we want to optimize share size. To exemplify this gap notice that the secret sharing scheme that results from the best formula for k-threshold on n parties has share size $\mathsf{poly}(\mathsf{k}) \cdot \log n$ [Fri86, Bop86], while the scheme of Shamir has size roughly $\log n$, independently of k.

shares to the parties such that any subset in $\mathcal{A}$ can reconstruct the secret from its shares, while any subset not in $\mathcal{A}$ cannot reveal any information on the secret.

More precisely, a standard secret sharing scheme for an access structure $\mathcal{A}$ consists of a pair of probabilistic algorithms $(\mathsf{SHARE}, \mathsf{RECON})$. SHARE gets as input a secret s and generates n shares $\Pi_1, \ldots, \Pi_n$. RECON gets as input shares of a subset B and outputs a string. The requirements are:

1. For every qualified set $B \in \mathcal{A}$, it holds that $\mathbf{Pr}[\mathsf{RECON}(\{\Pi_i\}_{i\in B}, B) = s] = 1$.
2. For every unqualified set $B \notin \mathcal{A}$ and every two different secrets s_1, s_2, it holds that the distributions $(\{\Pi_i\}_{i\in B}, s_1)$ and $(\{\Pi_i\}_{i\in B}, s_2)$ are identical.

The share size of a scheme is the maximum number of bits each party holds, namely $\max_{i\in n}\{|\Pi_i|\}$.

The well known scheme of Shamir [Sha79] for the (k, n)-THR access structure (based on polynomial interpolation) satisfies the following.

Claim 2.3 ([Sha79])**.** For every $n \in \mathbb{N}$ and $1 \leq k \leq n$, there is a secret sharing scheme for secrets of length m and the (k, n)-THR access structure in which the share size is ℓ, where $\ell \geq \max\{m, \log q\}$ and $q > n$ is a prime number (or a power of a prime). Moreover, if $k = 1$ or $k = n$, then $\ell = m$.[3]

2.1 Secret Sharing for Evolving Access Structures

We proceed with the definition of an evolving access structure. Roughly speaking, the parties arrive one by one and, in the most general case, a qualified subset is revealed only when all parties in that subset are present (in special cases the access structure is known to begin with, but there is no upper bound on the number of parties). To make sense of sharing a secret with respect to such a sequence of access structures, we require that the changes to the access structure are monotone, namely, parties are only added and qualified sets remain qualified.

To define evolving access structures we need to define a restriction.

Definition 2.4 (Restriction)**.** Let $\mathcal{A}$ be an access structure on n parties and let $0 < m < n$. We denote by $\mathcal{A}|_m$ the *restriction* of $\mathcal{A}$ to the first m parties. That is,

$$\mathcal{A}|_m = \{X \in \mathcal{A} \mid \forall i \in \{m+1, \ldots, n\}\colon\ i \notin X\}.$$

Due to monotonicity of the access structure, we have the following claim.

Claim 2.5. If $\mathcal{A}$ is an access structure on n parties, then $\mathcal{A}|_m$ is an access structure over m parties.

Proof. By definition of $\mathcal{A}|_m$, it contains only parties from the set $\{1, \ldots, m\}$. Thus, to prove the claim it is enough to show that $\mathcal{A}|_m$ is a monotone set, namely, that if $B \in \mathcal{A}|_m$ then for any $B \subseteq C \subseteq \mathcal{P}_m$. Indeed, since $\mathcal{A}$ is an access structure, for $B \in \mathcal{A}|_m$ and $B \subseteq C \subseteq \mathcal{P}_m \subseteq \mathcal{P}_n$, we have that $B, C \in \mathcal{A}$. By definition of $\mathcal{A}|_m$, it holds that $C \in \mathcal{A}|_m$. ■

Definition 2.6 (Evolving access structure)**.** A (possibly infinite) sequence of access structures $\{\mathcal{A}_t\}_{t\in\mathbb{N}}$ is called evolving if the following conditions hold

1. For every $t \in \mathbb{N}$, it holds that $\mathcal{A}_t$ is an access structure over t parties.

[3] Schemes in which the share size is equal to the secret size are known as *ideal* secret sharing schemes.

2. For every $t \in \mathbb{N}$, it holds that $\mathcal{A}_t|_{t-1}$ is equal to $\mathcal{A}_{t-1}$.[4]

This definition naturally gives rise to an evolving variant of threshold access structures (see Definition 2.2). Here, we think of k as fixed, namely, independent of the number of parties.

Definition 2.7 (Evolving threshold access structure)**.** For every $k \in \mathbb{N}$, let evolving k-THR be the evolving threshold access structure which contains for any access structure in the sequence all subsets of size at least k.

We generalize the definition of a standard secret sharing scheme to apply for evolving access structures. Intuitively, a secret sharing scheme for an evolving access structure is a secret sharing scheme in which the number of parties in not fixed in advance. At any point $t \in \mathbb{N}$ in time, there is an access structure $\mathcal{A}_t$ which defined the qualifies and unqualified subsets of parties.

Definition 2.8 (Secret sharing for evolving access structures)**.** Let $\mathcal{A} = \{\mathcal{A}_t\}_{t\in\mathbb{N}}$ be an evolving access structure. Let S be a domain of secrets. A secret sharing scheme for $\mathcal{A}$ and S consists of a pair of algorithms $(\mathsf{SHARE}, \mathsf{RECON})$. The sharing procedure SHARE and the reconstruction procedure RECON satisfy the following requirements:

1. $\mathsf{SHARE}(s, \{\Pi_1, \ldots, \Pi_{t-1}\})$ gets as input a secret $s \in S$ and the secret shares of parties $1, \ldots, t-1$. It outputs a share for the t^{th} party. For $t \in \mathbb{N}$ and secret shares $\Pi_1, \ldots, \Pi_{t-1}$ generated for parties $\{1, \ldots, t-1\}$, we let

$$\Pi_t \leftarrow \mathsf{SHARE}(s, \{\Pi_1, \ldots, \Pi_{t-1}\})$$

   be the secret share of party t.

   We abuse notation and sometimes denote by Π_t the random variable that corresponds to the secret share of party t generated as above.

2. **Correctness:** For every $t \in \mathbb{N}$, every qualified subset in $\mathcal{A}_t$ can reconstruct the secret. That is, for $t \in \mathbb{N}$ and $B \in \mathcal{A}_t$, it holds that

$$\mathbf{Pr}\left[\mathsf{RECON}(\{\Pi_i\}_{i\in B}, B) = s\right] = 1,$$

   where the probability is over the randomness of the sharing procedure.

3. **Secrecy:** For every $t \in \mathbb{N}$, every unqualified subset $B \notin \mathcal{A}_t$ and every two secret $s_1, s_2 \in S$, the distribution of the secret shares of parties in B generated with secret s_1 and the distribution of the shares of parties in B generated with secret s_2 are identical. Namely, the distributions $(\{\Pi_i\}_{i\in B}, s_1)$ and $(\{\Pi_i\}_{i\in B}, s_2)$ are identical.

The share size of the t^{th} party in a scheme for an evolving access structure is $|\Pi_t|$, namely the number of bits party t holds.

On choosing the access structure adaptively. One can also consider a stronger definition in which $\mathcal{A}_t$ is chosen at time t (rather than ahead of time) as long as the sequence of access structures $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_t\}$ is evolving. In this variant, the RECON procedure gets the access structure $\mathcal{A}_t$ as an additional parameter. Our construction of a secret sharing scheme for general evolving access structures in Section 4 works for this notion as well.

[4] Recall the definition of a *restriction* from Definition 2.4.

3 Warm-Up: Undirected S-T-Connectivity

We start with a simple warm-up scheme. We show that the standard scheme for the st-connectivity access structure can be easily adapted to the evolving setting.

In this access structure, parties correspond to edges of an *undirected* graph $G = (V, E)$. There are two fixed vertices in the graph called s and t (where $\mathsf{s}, \mathsf{t} \in V$). A set of parties (i.e., edges) is qualified if and only if they include a path from s to t.

Around 1989 Benaloh and Rudich [BR89] constructed a (standard) secret sharing for this access structure: let $s \in \{0,1\}$ be the secret. The dealer, given a secret s, assigns with each vertex $v \in V$ a label. For $v = \mathsf{s}$ the label is $w_{\mathsf{s}} = s$, for $v = \mathsf{t}$ the label is $w_{\mathsf{t}} = 0$ and for the rest of the vertices the label is chosen independently uniformly at random $w_v \leftarrow \{0,1\}$. The share of a party $e = (u, v) \in E$ is $w_u \oplus w_v$.

For correctness (completeness) consider a set of parties that include a path $\mathsf{s} = v_1 v_2 \dots v_k = \mathsf{t}$ from s to t. To reconstruct the secret, the parties XOR their shares to get

$$(w_{v_1} \oplus w_{v_2}) \oplus (w_{v_2} \oplus w_{v_3}) \oplus \cdots \oplus (w_{v_{k-1}} \oplus w_{v_k}) = w_{v_1} \oplus w_{v_k} = s.$$

For security, one can show that any subset of parties that do not form a path from s to t, hold shares which are independent of the secret. One way to prove this is to show that the number of ways to get to the shares of the parties given the secret 0 is equal to the number of ways to get to these shares given the secret 1 (see [Bei11, §3.2]).

One can observe that this access structure and scheme naturally generalize to the evolving setting. In this setting, we consider an evolving (possibly infinite) graph, where the set of nodes and edges are unbounded. At any point in time an arbitrary set of vertices and edges can be added to the graph. An addition of an edge corresponds to a new party added to the scheme. The special vertices s and t are fixed ahead of time and cannot change (this is to ensure the access structure is *evolving*).

Initially, the dealer assigns labels for the special vertices s and t, as before (i.e., it sets $w_{\mathsf{s}} = s$ and $w_{\mathsf{t}} = 0$). For the rest of the vertices the dealer assigns (uniformly random) labels only on demand: When a new edge $e = (u, v)$ is added to the graph (which corresponds to a new party), the dealer gives the party corresponding to the edge e the XOR of the labels of the vertices u and v. Correctness and security of this scheme follow similarly to the correctness and security of the standard scheme. One can see that the share size of each party is exactly the size of the secret.

4 A Scheme for General Evolving Access Structures

We give a construction of a secret sharing scheme for every evolving access structure. We emphasize that our construction also works in the scenario in which the access structure is chosen adaptively; see remark after Definition 2.8. Without loss of generality we focus on the case where the secret is a single bit.

Theorem 4.1 (Theorem 1.1 restated)**.** *For every evolving access structure there is a secret sharing scheme where the share size of the t^{th} party is at most 2^{t-1}.*

Our construction results with exponential-sized shares, that is, the share of the t^{th} party is 2^{t-1}. This should come as no surprise, as the best constructions known for standard secret sharing schemes for general access structures also have exponential-sized shares (exponential in the number

of parties). Proving that exponential-size shares are necessary to realize some *evolving* access structure is a very interesting open problem.

Our construction is influenced by a construction of Benaloh and Leichter [BL88] that gives a secret sharing scheme for any (standard) access structure $\mathcal{A}$ whose characteristic function $\mathbf{1}_{\mathcal{A}}$ is described by a monotone formula.[5] In this construction, each gate in the formula is assigned with a label (a bit). The labels assigned to the leaves correspond to the shares of the parties. Assign the root with the secret $s \in \{0,1\}$. Recursively apply the following assignment rules if a gate has an assigned label v and its children are not yet assigned: (1) if the gate is an $\vee$ gate, assign both its children with the label v, and (2) if the gate is an $\wedge$ gate, sample a random $u \leftarrow \{0,1\}$ and assign one child with the label u and the other with the label $v \oplus u$. The share of party i consists of all the labels of the leaves that correspond to the i^{th} input variable.

Reconstruction is done in the natural way from the leaves to the root. We assign a label to each gate. The label of a leaf corresponding to the i^{th} input variable is the corresponding share of the i^{th} party. The label of an $\wedge$ gate is the XOR of the labels of its two children. The label of an $\vee$ gate is the label of either of its children (by construction they are the same). The label of the root is the output of the reconstruction.

Correctness and security of the scheme follow by induction on the depth of the formula. For formulas of depth 1 (i.e., formulas that contain a single leaf), correctness and security are immediate. Consider a formula F of depth $d > 0$ and assume that the scheme is correct and secure for formulas of depth $d-1$. The formula F can be written as either $F = F_1 \vee F_2$ or $F = F_1 \wedge F_2$, where F_1 and F_2 are the left and right subtrees, respectively. Let x and y be inputs to the formula F that corresponds to a qualified and unqualified set of parties, respectively. That is $F(x) = 1$ and $F(y) = 0$. There are two cases to analyze:

1. If $F = F_1 \vee F_2$: Either $F_1(x) = 1$ or $F_2(x) = 1$ (or both). Thus, by the hypothesis, one of the labels of the children of F can be learned from the shares of labels of x. By construction, this label is also the label of F. For security notice that both $F_1(y) = 0$ and $F_2(y) = 0$. Thus, by hypothesis, both labels of the children of F cannot be learned from the shares of y. Therefore, by construction, the label of the root F is perfectly hidden,

2. If $F = F_1 \wedge F_2$: Both $F_1(x) = 1$ and $F_2(x) = 1$. Thus, by the hypothesis, both labels of the children of F can be learned from the shares of labels of x. By construction, one can recover the label of F by summing up both labels. For security notice that either $F_1(y) = 0$ or $F_1(y) = 0$ (or both). Thus, by hypothesis, at least one label of the children of F cannot be learned from the shares of y. Therefore, by construction, the label of the root F is perfectly hidden.

The share size of each party is determined by the number of times the corresponding input variable appears as a leaf in the formula.

Proof of Theorem 4.1. Our construction exploits the following fact about monotone functions: For every monotone function $f\colon \{0,1\}^n \to \{0,1\}$ it holds that

$$f(x_1,\ldots,x_n) = (x_n \ \wedge \ f(x_1,\ldots,x_{n-1},1)) \ \vee \ f(x_1,\ldots,x_{n-1},0). \tag{4.1}$$

[5] A monotone formula is a binary tree in which every leaf is labeled by an input variable and every internal node is labeled by an $\wedge$ or $\vee$ gate. The root of the tree is the output gate. Recall that an access structure is a monotone set of subsets. Thus, there must be a monotone formula that computes its characteristic function.

Using this fact recursively, we can write any function f as a formula F which we call the normal form formula of f. Notice that $f(x_1, \ldots, x_{n-1}, 0)$ and $f(x_1, \ldots, x_{n-1}, 1)$ are independent of x_n. Our construction will apply a variant of the [BL88] construction. Recall that in the standard setting, secret sharing according to a monotone formula with constant leaves (fixed in $\{0,1\}$) is not a problem since any such formula can be *simplified* to contain only (unfixed) input variables. Our normal form formula, on the other hand, contains constants that we cannot simplify (due to the uncertainty regarding the future). Thus, we handle constants in a special way, similarly to how we handle input variables. In particular, the dealer will have to remember the labels of the constant leaves. Details follow.

Let $\mathcal{A} = \{\mathcal{A}_t\}_{t\in\mathbb{N}}$ be an evolving access structure.[6] Let $\{f_t\}_{t\in\mathbb{N}}$ be the sequence of functions, where $f_i\colon \{0,1\}^i \to \{0,1\}$ is the (monotone) characteristic function of $\mathcal{A}_i$. The dealer maintains a labeled tree representing the formula in normal form, while updating the tree with each new party arriving. At time step $t = 1$ the dealer secret shares the secret with respect to the access structure f_1 according to the formula

$$f_1(x_1) = (x_1 \ \wedge \ f_1(1)) \ \vee \ f_1(0).$$

That is, the dealer assigns the leaf $f_1(0)$ with the label s, samples a random bit $r_{(1)} \leftarrow \{0,1\}$, assigns x_1 with the label $r_{(1)}$ and $f^{(1)}(1)$ with the label $r_{(1)} \oplus s$. The share of party 1 is s if $f_1(1) = 1$ and it is $r_{(1)}$ otherwise. The dealer remembers the tree with the above labels depicted in Figure 1.

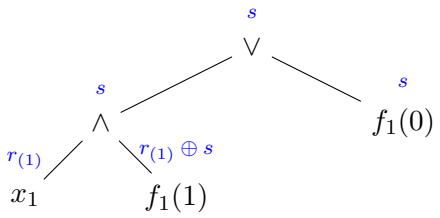

Figure 1: The tree corresponding to the formula $f_1(x_1)$ in normal form.

When party $t > 1$ arrives, the dealer, that maintains a normal form formula for f_{t-1} with corresponding labels on the gates, proceeds as follows. For every $\vec{b} = (b_1, \ldots, b_{t-1}) \in \{0,1\}^{t-1}$, it expands the constant gate $f_{t-1}(b_1, \ldots, b_{t-1})$ labeled with $r_{\vec{b}} \oplus s$ with the subtree

$$(x_t \ \wedge \ f_t(b_1, \ldots, b_{t-1}, 1)) \ \vee \ f_t(b_1, \ldots, b_{t-1}, 0).$$

Next, it samples a random bit $\mathsf{b}_{\vec{b}} \leftarrow \{0,1\}$ and labels the leaf x_t with $\mathsf{b}_{\vec{b}}$, the leaf $f_t(b_1, \ldots, b_{t-1}, 1)$ with $\mathsf{b}_{\vec{b}} \oplus r_{\vec{b}} \oplus s$ and the leaf $f_t(b_1, \ldots, b_{t-1}, 0)$ with $r_{\vec{b}} \oplus s$, respectively. Finally, the bit $r_{\vec{b}} \oplus s$ is given to the t^{th} party if $f_t(b_1, \ldots, b_{t-1}, 1) = 1$. Otherwise, the bit $\mathsf{b}_{\vec{b}}$ is given to party t. This is depicted in Figure 2.

[6] Our construction will actually work in the setting where $\mathcal{A}_t$ itself is chosen at time t (and it is not known at time $t' < t$).

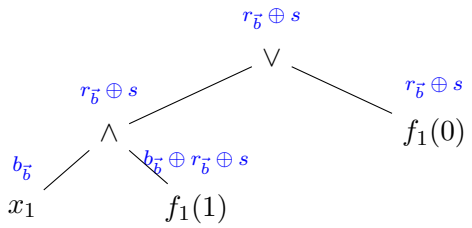

Figure 2: The tree corresponding to the formula $f_t(\vec{b})$ for $\vec{b} \in \{0,1\}^{t-1}$ in normal form.

For example, when party 2 arrives, we expand the leaves that are constants in Figure 1 (i.e., $f_1(0)$ and $f_1(1)$) with the appropriate subtree. Within each subtree, we apply the secret sharing scheme described above where the secret is the label of the root (it is $r_{(1)} \oplus s$ for $f_1(1)$ and s for $f_1(0)$). The resulting formula is depicted in Figure 3. Party 2 gets the bit $r_{(1)} \oplus s$ if $f_2(1,1) = 1$, and it gets the bit $r_{(1,1)}$ otherwise. In addition, party 2 gets the bit s if $f_2(0,1) = 1$, and it gets $r_{(0,1)}$ otherwise.

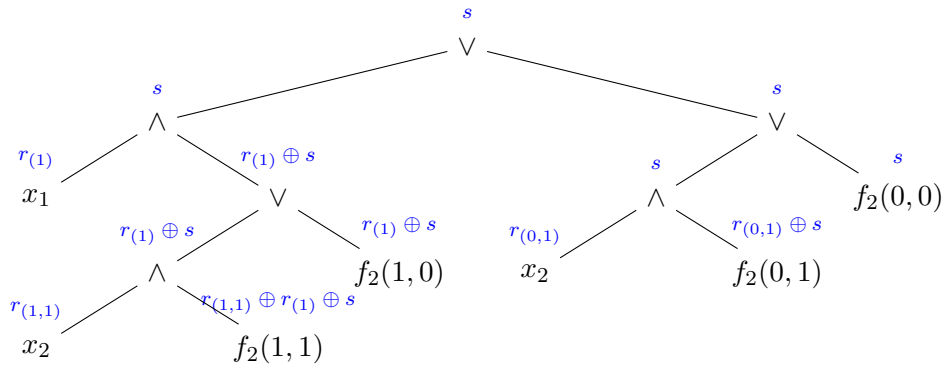

Figure 3: The tree corresponding to the formula $f_2(x_1, x_2)$ in normal form.

Reconstruction at time t is done bottom up from the leaves to the root according of the normal form formula of f_t. The leaves that correspond to input variables are labeled using the shares of the corresponding parties in the natural way. The label of an $\wedge$ gate whose children are labeled, is the XOR of the labels of its children. The label of an $\vee$ gate whose children are labeled, is the label of either of its children (they are the same by construction).

Next, we argue correctness and security at time $t \in \mathbb{N}$. The following claim, that follows from the construction, will be useful.

Claim 4.2. At time $t \in \mathbb{N}$, for every $\vec{b} = (b_1, \ldots, b_{t-1}) \in \{0,1\}^t$ it holds that

$$r_{\vec{b}} = \left(\mathsf{b}_{(b_1,\ldots,b_{t-1})} \wedge b_t\right) \oplus \left(\mathsf{b}_{(b_1,\ldots,b_{t-2})} \wedge b_{t-2}\right) \oplus \cdots \oplus \left(\mathsf{b}_{(b_1)} \wedge b_1\right).$$

Let $\vec{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$ be an indicator vector of a qualified set of parties at time t. Let us assume without loss of generality that $b_t = 1$. For every $i \in [t-1]$ such that $b_i = 1$, party i holds the bit $\mathsf{b}_{(b_1,\ldots,b_i)}$. Party t, by construction and using Claim 4.2, holds the bit

$$\left(\mathsf{b}_{(b_1,\ldots,b_{t-1})} \wedge b_{t-1}\right) \oplus \cdots \oplus \left(\mathsf{b}_{(b_1)} \wedge b_1\right) \oplus s.$$

Therefore, by XOR-ing all the above shares the parties can compute s.

Security is immediate for $t = 1$. Assume that the scheme is secure for $t-1$ and we shall prove that the scheme is secure also for $1 < t \in \mathbb{N}$. Let $\vec{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$ be an indicator vector of an unqualified set of parties at time t. Assume that $b_t = 1$, as the other case follows immediately from the induction hypothesis. For every $\vec{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$, party t receives either a uniform random bit $\mathsf{b}_{(b_1,\ldots,b_t)}$ or the bit $\left(\mathsf{b}_{(b_1,\ldots,b_{t-1})} \wedge b_{t-1}\right) \oplus \cdots \oplus \left(\mathsf{b}_{(b_1)} \wedge b_1\right) \oplus s$, depending on whether $f_t(b_1, \ldots, b_t) = 0$ or not. The $\mathsf{b}_{(b_1,\ldots,b_t)}$ bits do not give an unqualified set any additional information about the secret as they are independent of everything else this set posses, so we can ignore them. For any bit of the form $\left(\mathsf{b}_{(b_1,\ldots,b_{t-1})} \wedge b_{t-1}\right) \oplus \cdots \oplus \left(\mathsf{b}_{(b_1)} \wedge b_1\right) \oplus s$, by construction any unqualified set is missing at least one of the XORed components $(\mathsf{b}_{(b_1,\ldots,b_i)} \wedge b_i)$ for which $b_i = 1$ (as otherwise it must be qualified). Therefore, these bits are also distributed independently uniformly at random and do not give any information about s. Security follows by the hypothesis.

Share size. Each leaf of the tree contributes one bit to the share of the corresponding party. By construction, the total size of a share of a party is equal to the number of leaves which are labeled by constants in the round before that party arrived. The number of such leaves (i.e., leaves labeled by constants) doubles every round. The first party gets a single bit and has a tree with two constant nodes. The second party thus gets two bits and has a tree with four constant nodes, and so on. In general, we have that the share size of party t is 2^{t-1} bits.

Information the dealer maintains. To prepare a share for party $t \in \mathbb{N}$, the dealer has to maintain the 2^{t-1} labels (each is one bit) corresponding to the constant leaves $f_{t-1}(b_1 \ldots, b_{t-1})$ for $b_1, \ldots, b_{t-1} \in \{0,1\}$.

Remark 4.3 (Efficiency improvements.)**.** It is possible to reduce the share size of some parties by slightly optimizing the above scheme. Whenever a constant leaf $f_t(b_1, \ldots, b_t)$ for $t \in \mathbb{N}$ and $b_1 \ldots, b_t \in \{0,1\}$ that evaluates to 1 is encountered, we do not need to expand it in future time steps. Indeed, the fact that $f_t(b_1, \ldots, b_t) = 1$ implies that the set that $(b_1, \ldots, b_t)$ represents ($b_i = 1$ represents that party $i \in [t]$ is present and vice versa) is qualified. Any subset that contains parties that came later than time step t and include this subset, can recover the secret by using the shares of parties that $(b_1, \ldots, b_t)$ represents (ignoring the shares of parties that come later).

■

Generalization to larger domains of secrets. One can generalize the scheme to support larger domains of secrets. One way to do this is by secret sharing long secrets bit-by-bit (assuming a representation of secrets as bits). Another way to do this is to work over a larger finite group G. Here, sampling random elements is done by sampling independently with uniform distribution over the elements of G. The XOR operation should be replaced with the group operation of G.

5 An Efficient Scheme for Evolving 2-Threshold

In this section we give an efficient construction for a secret sharing scheme for the evolving 2-THR access structure. Recall that evolving 2-THR is the sequence of access structures $(2,1)$-THR, $(2,2)$-THR, $(2,3)$-THR, ... which allow, at any point in time, for every pair of parties to learn the secret while disallowing singletons to learn anything about it. We focus on the case where the secret is a single bit.

Theorem 5.1. *There is a secret sharing scheme for the evolving* 2-THR *access structure in which the share size of the t^{th} party is bounded by*

$$\log t + \log\log t + 2\log\log\log t + 6.$$

Recall that in the classical setting of secret sharing, where an upper bound on the number of parties is known, there is a very efficient scheme for $(2, n)$-THR in which the share size of each party is roughly $\log n$ (see Claim 2.3). In Section 7 we show that in the evolving setting, for any $c \in \mathbb{N}$, a scheme in which the share size of the t^{th} party is $\log t + \log\log t + c$ cannot exist. Thus, up to an additive $\log\log\log t$ term, our scheme is optimal.

Our main technical claim used to prove Theorem 5.1 is given in the following lemma.

Lemma 5.2. *Assume that there exists a secret sharing scheme for the evolving* 2-THR *access structure in which the share size of the t^{th} party is $\sigma(t)$. Then, there exists a secret sharing scheme for the evolving* 2-THR *access structure in which the share size of the t^{th} party is*

$$\log t + \sigma(\log t + 1).$$

Proof of Theorem 5.1 assuming Lemma 5.2. Recall that in Section 4 we constructed a secret sharing scheme for any evolving access structure. By applying it we would get shares of size 2^{t-1}. However, using the efficiency improvements described in Remark 4.3, we can get a better bound[7]. Consider the tree at time $t-1$. The only leaf corresponding to constants that are 0 are of the form $f_{t-1}(b_1, \ldots, b_{t-1})$ where $\sum_{i=1}^{t-1} b_i \leq 1$. There are exactly t such leaves, and thus we get a scheme $\Pi^{(0)}$ for evolving 2-THR in which the share size of the t^{th} party is

$$\sigma^{(0)}(t) = t.$$

Using Lemma 5.2 this gives rise to a scheme $\Pi^{(1)}$ in which the share size of the t^{th} party is

$$\begin{aligned}\sigma^{(1)}(t) &= \log t + \sigma^{(0)}(\log t + 1)\\ &= 2\log t + 1.\end{aligned}$$

Applying Lemma 5.2 again we get a scheme $\Pi^{(2)}$ in which the share size of the t^{th} party is

$$\begin{aligned}\sigma^{(2)}(t) &= \log t + \sigma^{(1)}(\log t + 1)\\ &\leq \log t + 2\log(\log t + 1) + 1\\ &\leq \log t + 2\log\log t + 3.\end{aligned}$$

Applying Lemma 5.2 one last time we get a scheme $\Pi^{(3)}$ in which the share size of the t^{th} party is

$$\begin{aligned}\sigma^{(3)}(t) &= \log t + \sigma^{(2)}(\log t + 1)\\ &\leq \log t + \log(\log t + 1) + 2\log\log(\log t + 1) + 3\\ &\leq \log t + \log\log t + 2\log\log\log t + 6.\end{aligned}$$

This proves the theorem.

We note that this bound is tight according to the lower bound in Theorem 1.3 up to the low-order term $\log\log\log t$.

We note that by applying Lemma 5.2 i times we can improve the share size for large enough t. This will match the lower bound up to low order term of $\log^{(i)}(t)$ (See Remark 7.3). We choose to stop after three applications of Lemma 5.2 due to aesthetic reasons. ∎

[7] Alternatively, we can use the construction of [CT12] (see Section 1.3) which gives the t^{th} party a share of size t.

We are left to prove Lemma 5.2.

Proof of Lemma 5.2. Let Π be a construction of a secret sharing scheme for evolving 2-THR in which the share size of the t^{th} party is $\sigma(t)$. We construct a scheme Π' for the same access structure in which the share size is $\log t + \sigma(\log t + 1)$. We proceed with the description of the scheme.

Let $s \in \{0,1\}$ be the secret to be shared. Each party, whenever it arrives, is assigned to a generation. The generations are growing in size: For $g = 0, 1, 2 \ldots$ the g^{th} generation begins when the 2^g-th party arrives. Therefore, the size of the g^{th} generation, namely, the number of parties that are part of this generation, is $\text{SIZE}(g) = 2^g$ and party $t \in \mathbb{N}$ is part of generation $g = \lfloor \log t \rfloor$.

Whenever a generation begins the dealer prepares shares for all parties that are part of that generation. Let us focus on the beginning of the g^{th} generation and describe the dealer's procedure:

1. Split s using a secret sharing scheme for $(2, \text{SIZE}(g))$-THR. Denote the resulting shares by $u_1^{(g)}, \ldots, u_{\text{SIZE}(g)}^{(g)}$.
2. Generate one share using the secret sharing scheme Π given the secret s and previous shares $\{v^{(i)}\}_{i \in \{0,\ldots,g-1\}}$. Denote the resulting share by $v^{(g)}$.
3. Set the secret share of the j^{th} party in the g^{th} generation (i.e., $j \in [\text{SIZE}(g)]$) to be
$$\left(u_j^{(g)}, v^{(g)}\right).$$

The output of the scheme is depicted in Figure 4.

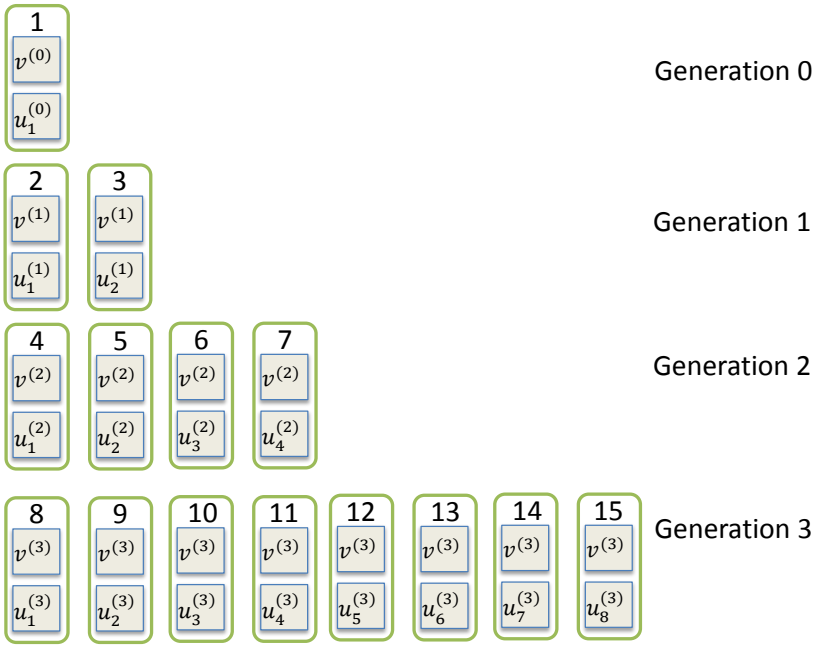

Figure 4: The shares of parties $1, \ldots, 15$ from generations $0, \ldots, 3$.

Correctness and security. Let $t_1, t_2 \in \mathbb{N}$ be any two different parties. We show that the secret s can be computed from their shares. If t_1 and t_2 are from the same generation g (i.e., if $g = \lfloor \log t_1 \rfloor = \lfloor \log t_2 \rfloor$), then they can reconstruct the secret s using the reconstruction procedure of the $(2, \text{SIZE}(g))$-THR scheme using the corresponding $u^{(g)}$ shares. If they are from different generations $g_1 \neq g_2$, then the parties can compute s using the reconstruction procedure of the evolving 2-THR scheme and the two shares $v^{(g_1)}$ and $v^{(g_2)}$.

For security consider any single party $t \in \mathbb{N}$ from generation g. By the security of the $(2, \text{SIZE}(g))$-THR scheme, the security of the evolving 2-THR scheme, and the fact that both parts of the share are generated independently, the shares cannot be used to learn anything about the secret.

Share size analysis. We analyze the share size of parties in the scheme Π'. Denote by $\sigma(t)$ the share size of party t in the scheme Π. We bound the size of each component in the share of party t. The share of party t that is the j^{th} party of generation $g = \lfloor \log t \rfloor$ is $(u_j^{(g)}, v^{(g)})$.

1. $u_j^{(g)}$ – generated by secret sharing s using a scheme for $(2, \text{SIZE}(g))$-THR. Since $\text{SIZE}(g) = 2^g$ and using Claim 2.3 we get that

$$|u_j^{(g)}| \leq \log\left(\text{SIZE}(g)\right) \leq \lfloor \log t \rfloor.$$

2. $v^{(g)}$ – generated by generating one share of a secret sharing scheme Π for evolving 2-THR. Recall that g shares were generated for previous generations. Therefore,

$$|v^{(g)}| = \sigma(g+1) = \sigma(\lfloor \log t \rfloor + 1).$$

Thus, the total share size in the scheme Π' is bounded by

$$\log t + \sigma(\log t + 1).$$

■

Generalization to larger domains of secrets. As we mentioned in Section 4, the scheme that follows from Theorem 1.1 can be used to share longer secrets (bit by bit). Shamir's threshold scheme, on the other hand, can be used to share a longer secret than 1 bit without increasing the share size (it can be used to share a secret of size roughly log number of parties in the scheme). Using these features our scheme can be generalized to support sharing longer secrets without paying too much in share size.

6 A Scheme for Evolving k-Threshold

In this section we give a construction for a secret sharing scheme for the evolving k-THR access structure. Again, we focus on the case where the secret is a single bit.

Theorem 6.1 (Theorem 1.2 restated)**.** *There is a secret sharing scheme for the evolving k-THR access structure in which the share size of the t^{th} party is at most*

$$(k-1) \cdot \log t + 6k^3 \cdot \log\log t \cdot \log\log\log t + 7k^4 \cdot \log k.$$

As in the case of $k = 2$ (see the discussion after Theorem 5.1), the best one could hope to obtain is a scheme in which the share of the t^{th} party is close to $\log t$.[8] Our construction has a linear dependence on k and we leave open the question whether this can be improved.

We note that the bound in Theorem 6.1 applies for any $t \in \mathbb{N}$ and $k \geq 2$. For specific values of t and k one can follow the analysis and obtain a much better bound.

Our main technical lemma used to prove Theorem 6.1 is a general transformation where we take any scheme for the evolving k-THR access structure (possibly with large share size), and convert it into a different scheme with smaller share size. Formally we prove following lemma.

Lemma 6.2. *Let $k \in \mathbb{N}$. Assume that there exists a secret sharing scheme for the evolving k-THR access structure in which the share size of the t^{th} party is $\sigma(t)$. Then, there exists a secret sharing scheme for the evolving k-THR access structure in which the share size of the t^{th} party is at most*

$$(k-1)\log t + k \cdot \sigma(\log t + k) + k^2.$$

The proof of Theorem 6.1 is done via repeated applications of Lemma 6.2, somewhat similarly to the proof of Theorem 5.1. However, naively the resulting parameters are not very good. Specifically, if we start with the scheme for evolving k-THR in which the share size is exponential (which is what we get using the scheme from Theorem 1.1), then by applying Lemma 6.2, the share size will eventually depend *exponentially* on k.

To overcome this, we first present a tailor-made construction for evolving k-THR in which the share size of party t has *almost linear* dependence on t and k. Using this scheme as a basic building block, we repeatedly apply Lemma 6.2 to obtain Theorem 6.1. The proof of the latter can be found in Section 6.3. The tailor-made construction for evolving k-THR appears next in Section 6.1. Finally, the proof of Lemma 6.2 appears in Section 6.2.

6.1 The basic scheme for evolving k-threshold

The main result of this subsection is a construction for evolving k-THR in which the share size of party t is almost linear in t and k. This scheme will be used later as the basic building block in our final scheme for evolving k-THR satisfying Theorem 6.1.

Lemma 6.3. *There is a secret sharing scheme for the evolving k-THR access structure in which the share size of the t^{th} party is bounded by $kt \cdot \log(kt)$.*

To prove Lemma 6.3, we construct a secret sharing scheme for a new access structure. The access structure C_ℓ over $2k$ parties is defined via its characteristic monotone function that we denote by C_ℓ as well. Let $C_\ell \colon \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ be the function that on input $(x, y) \in \{0,1\}^k \times \{0,1\}^k$. Roughly speaking, we think of x and y as unary encoding of two numbers in $\{0, \ldots, k\}$.[9] The access structure contains all pairs whose sum is at least ℓ. Formally, we define $C_\ell(x, y) = 1$ if and only if at least one of the following conditions hold:

1. $\exists i, j \in [\ell - 1]$ such that $x_i = 1$, $y_j = 1$ and $i + j = \ell$.
2. $y_\ell = 1$ or $x_\ell = 1$.

[8] Shamir's scheme for (k, n)-THR results with shares of size roughly $\log n$. In particular, independent of k.

[9] The variable x will represent the number of parties present so far and y will represent the number of parties to come. The role of both inputs will become clear later in this section.

Claim 6.4. Let $\ell, k \in \mathbb{N}$. There exists a secret sharing scheme for the access structure C_ℓ in which the share size of each party is exactly the size of the shared secret.

Proof. The following formula computes C_ℓ:

$$C_\ell(x,y) = \bigvee_{i=1}^{\ell-1} (x_i \wedge y_{\ell-i}) \vee (x_\ell \vee y_\ell)$$

Notice that this formula is a DNF and every input variable appears exactly once. This formula gives rise to a simple secret sharing scheme for the access structure C_ℓ using the method of [BL88] (see Section 4 for the description of the scheme). Since each variable appears at most once in the formula, the share of each party is at most exactly the length of the secret. The theorem follows by padding all shares to be of the same length. ■

Proof of Lemma 6.3. We construct a scheme Π_k for the evolving k-THR access structure in which the share size is $kt \cdot \log(kt)$.

Let $s \in \{0,1\}$ be the secret to be shared. Each party, whenever it arrives, is assigned to a generation. Party $t \in \mathbb{N}$ is assigned to generation $g = \lfloor \log_k t \rfloor$. The generations are growing in size: For $g = 0, 1, 2 \ldots$ the g^{th} generation begins when the k^g-th party arrives. Therefore, the size of the g^{th} generation, (the number of parties that are part of this generation) is

$$\text{SIZE}(g) = k^{g+1} - k^g = (k-1) \cdot k^g.$$

Whenever a generation begins the dealer prepares shares for all parties that are part of that generation. For each generation g the dealer generates, in addition to the shares that it distributes to the parties of that generation, k^{g+1} strings $\{y^{(g+1)}_{\vec{z}}\}_{\vec{z} \in [k]^{g+1}}$ which it remembers for the next generation. Initially, set $y^{(0)}_{(k)} = s$. Let us focus on the beginning of the g^{th} generation and describe the dealer's procedure (we define $[k]^0 = \varnothing$):

1. For all $\vec{z} = (i_0, \ldots, i_g) \in \{k\} \times [k]^g$ split the string $y^{(g)}_{\vec{z}}$ using a secret sharing scheme for C_{i_g}. Denote the resulting shares by $x^{(g)}_{(\vec{z},1)}, \ldots, x^{(g)}_{(\vec{z},k)}, y^{(g+1)}_{(\vec{z},1)}, \ldots, y^{(g+1)}_{(\vec{z},k)}$.

2. For all $\vec{z} = (i_0, \ldots, i_{g+1}) \in \{k\} \times [k]^{g+1}$ secret share $x^{(g)}_{\vec{z}}$ using a scheme for $(i_{g+1}, \text{SIZE}(g))$-THR. Denote the resulting shares by $u^{(g)}_{\vec{z},1}, \ldots, u^{(g)}_{\vec{z},\text{SIZE}(g)}$.

3. For any $j \in [\text{SIZE}(g)]$, set the secret share of the j^{th} party in the g^{th} generation to be

$$\{u^{(g)}_{\vec{z},j}\}_{\vec{z} \in \{k\} \times [k]^{g+1}}.$$

Correctness and security. We start with the following claim.

Claim 6.5. Any $c \in [\text{SIZE}(g)]$ parties from generation g can compute $\{x^{(g)}_{(\vec{z},i)}\}_{\vec{z} \in \{k\} \times [k]^g, i \in [c]}$.

Proof. Let $j_1, \ldots, j_c \in [\text{SIZE}(g)]$ be the indices of parties present from that generation. Thus, the parties can compute

$$\{u^{(g)}_{\vec{z},j_1}, \ldots, u^{(g)}_{\vec{z},j_c}\}_{\vec{z} \in \{k\} \times [k]^{g+1}}.$$

Therefore, all the x values that were shared via a threshold scheme in which the threshold was at most c can be reconstructed. Namely, the values $\{x^{(g)}_{(\vec{z},i)}\}_{\vec{z} \in \{k\} \times [k]^g, i \in [c]}$. ■

Claim 6.6. Fix a generation $g \in \mathbb{N}$, two number $c_1, c_2 \in [k]$ and $z = (i_0, \ldots, i_g) \in \{k\} \times [k]^g$. Then, given $x^{(g)}_{(\vec{z},c_1)}$ and $y^{(g+1)}_{(\vec{z},c_2)}$ such that $c_1 + c_2 \geq i_g$, one can compute $y^{(g)}_{\vec{z}}$.

Moreover, given $x^{(g)}_{(\vec{z},c_1)}$ such that $c_1 \geq i_g$, one can compute $y^{(g)}_{\vec{z}}$.

Proof. Follows from the correctness of the secret sharing scheme for C_ℓ. ■

Now, let us assume that k parties come together and try to reconstruct s. Assume that c_0 parties come from generation 0, c_1 come from generation 1 and so on. That is, for some generation g it holds that $\sum_{i=0}^{g} c_i = k$ and without loss of generality $c_g > 0$. We show that these parties can learn $y^{(0)}_{(k)}$ which is s by construction. This is done by applying Claims 6.5 and 6.6 alternately and iteratively. Details follow.

By Claim 6.5, using the shares of the c_g parties in the last generation we can compute $\{x^{(g)}_{(\vec{z},c_g)}\}_{\vec{z}\in\{k\}\times[k]^g}$. By Claim 6.6, using these shares we can reconstruct $\{y^{(g)}_{\vec{z}}\}_{\vec{z}=(i_0,\ldots,i_g)\in\{k\}\times[k]^{g-1}\times\{c_g\}}$. Using Claim 6.5 again we can compute $\{x^{(g-1)}_{(\vec{z},c_{g-1})}\}_{\vec{z}\in\{k\}\times[k]^{g-1}}$, using Claim 6.6 combine these shares with $\{y^{(g)}_{\vec{z}}\}_{\vec{z}=(i_0,\ldots,i_g)\in\{k\}\times[k]^{g-1}\times\{c_g\}}$ and reconstruct $\{y^{(g-1)}_{\vec{z}}\}_{\vec{z}=(i_0,\ldots,i_{g-1})\in\{k\}\times[k]^{g-2}\times\{c_g+c_{g-1}\}}$. Applying this argument iteratively one can eventually compute $y^{(0)}_{(k)} = s$, as required.

For security fix any set of parties as above where $\sum_{i=0}^{g} c_i < k$. We argue that these parties cannot learn the value $y^{(0)}_{k}$. From the security of the scheme for C_ℓ, it is enough to show that they cannot learn any value in $\{y^{(1)}_{(\vec{z},k-c_0)}\}_{\vec{z}\in\{k\}\times[k]}$. Applying this logic once again, it is enough to show that they cannot learn any value in $\{y^{(2)}_{(\vec{z},k-c_0-c_1)}\}_{\vec{z}\in\{k\}\times[k]^2}$. Applying this argument g times, we get that s cannot be learned if and only if $\{y^{(g+1)}_{(\vec{z},k-\sum_{i=1}^{g} c_i)}\}_{\vec{z}\in\{k\}\times[k]^g}$ cannot be learned. Indeed, these values are independent of the shares of parties up to generation g.

Share size analysis. We analyze the share size of parties in the scheme Π_k described above. The share of party t from generation g is composed of k^{g+1} shares generated via standard threshold schemes over $\textsc{size}(g)$ parties. Thus, in total, the share size of party t is bounded by $k^{g+1} \cdot \log(\textsc{size}(g))$. Recall that $g = \lfloor \log_k t \rfloor$ and $\textsc{size}(g) = (k-1) \cdot k^g$. Therefore, the share size is bounded by

$$k \cdot t \cdot \log((k-1) \cdot t) \leq kt \cdot \log(kt).$$

■

6.2 Recursive composition: Proof of Lemma 6.2

Let Π_k be a construction of a secret sharing scheme for evolving k-THR in which the share size of the t^{th} party is $\sigma_k(t)$. We construct a scheme Π'_k for the same access structure in which the share size is $\sigma'_k(t) = \log t + (k-1) + \sigma(\log t + (k-1)) + (k-2) \cdot \max\{\log t + (k-1), \sigma(\log t + (k-1))\}$.

Let $s \in \{0,1\}$ be the secret to be shared. Each party, whenever it arrives, is assigned to a generation. The generations are growing in size: For $g = 0, 1, 2 \ldots$ the g^{th} generation begins when the $2^{(k-1)\cdot g}$-th party arrives. Thus, party $t \in \mathbb{N}$ is part of generation $g = \lfloor (\log t)/(k-1) \rfloor$, and the number of parties that are part of generation g, is

$$\textsc{size}(g) = 2^{(k-1)\cdot(g+1)} - 2^{(k-1)\cdot g} = 2^{(k-1)\cdot g} \cdot (2^{k-1} - 1) \leq t \cdot 2^{k-1}.$$

Whenever a generation begins the dealer prepares shares for all parties that are part of that generation. We focus on the beginning of generation g and describe the dealer's procedure:

1. Split s using a secret sharing scheme for $(k, \text{SIZE}(g))$-THR. Denote the resulting shares by $u_1^{(g)}, \ldots, u_{\text{SIZE}(g)}^{(g)}$.
2. Generate $k-1$ shares using the secret sharing scheme Π_k given the secret s and previous shares $\{v_j^{(i)}\}_{i\in[g-1],j\in[k-1]}$. Denote the resulting shares by $v_1^{(g)}, \ldots, v_{k-1}^{(g)}$.
3. For $i \in [k-1]$, split $v_i^{(g)}$ using a secret sharing scheme for $(i, \text{SIZE}(g))$-THR. Denote the resulting shares by $\{w_{i,1}^{(g)}, \ldots, w_{i,\text{SIZE}(g)}^{(g)}\}_{i\in[k-1]}$.
4. Set the secret share of the j^{th} party in the g^{th} generation (i.e., $j \in [\text{SIZE}(g)]$) to be

$$\left(u_j^{(g)}, w_{1,j}^{(g)}, \ldots, w_{k-1,j}^{(g)}\right).$$

Correctness and security. We show that any k parties can learn the secret. If all the parties come from the same generation g, then they can use their $u^{(g)}$ in order to run the reconstruction procedure of the $(k, \text{SIZE}(g))$-THR scheme and learn s. For k parties that come from at least two generations we show that they can jointly learn k shares for the evolving k-THR scheme Π_k. By correctness of Π_k, using these shares they can reconstruct s. Indeed, assume that c_0 parties come from generation 0, c_1 come from generation 1 and so on, where there is some generation g where $\sum_{i=0}^{g} c_i = k$ and for all i it holds that $c_i \leq k-1$.

Claim 6.7. Any $c \in [\text{SIZE}(g)]$ parties from generation g can compute $v_c^{(g)}$.

Proof. The c parties hold c shares for $(1, \text{SIZE}(g))$-THR scheme that give $v_1^{(g)}$, c shares for the $(2, \text{SIZE}(g))$-THR scheme that give $v_2^{(g)}$ and so on. ■

Using this claim we get that the k parties can learn $\sum_{i=0}^{g} c_i = k$ shares of the evolving k-THR scheme, as required.

For security consider any set of $k-1$ parties. First, the u shares of the $(k, \text{SIZE}(g))$-THR scheme are independent of the secret. Thus, to complete the proof we need to show that the parties cannot learn any k shares of the evolving k-THR scheme Π_k. Indeed, any c parties from generation g cannot learn more than c shares $v_1^{(g)}, \ldots, v_c^{(g)}$; this follows from the security of the schemes $(c+1, \text{SIZE}(g))$-THR, $\ldots$, $(k-1, \text{SIZE}(g))$-THR. Therefore, in total, the parties can learn at most $\sum_{i=0}^{g} c_i < k$ shares.

Share size analysis. We bound the size of each component in the share of party t in the scheme Π'_k. The share of party t that is the j^{th} party of generation $g = \lfloor (\log t)/(k-1) \rfloor$ is composed of $u_j^{(g)}$ and $w_{1,j}^{(g)}, \ldots, w_{k-1,j}^{(g)}$:

1. $u_j^{(g)}$ – generated by secret sharing s using a scheme for $(k, \text{SIZE}(g))$-THR. By Claim 2.3 it holds that

$$|u_j^{(g)}| \leq \log(\text{SIZE}(g)) \leq \log t + (k-1)$$

2. $w_{i,j}^{(g)}$ – generated by secret sharing $v_i^{(g)}$ using a scheme for $(i, \text{SIZE}(g))$-THR. By Claim 2.3 for $1 < i \leq k-1$ it holds that

$$|w_{i,j}^{(g)}| \leq \max\{\log(\text{SIZE}(g)), |v_i^{(g)}|\} \leq \max\{\log t + (k-1), |v_i^{(g)}|\}$$

and for $i = 1$ it holds that

$$|w_{1,j}^{(g)}| = |v_i^{(g)}|.$$

- $v_i^{(g)}$ – generated by generating a share of a sharing scheme Π_k for evolving k-THR. Recall that $g \cdot (k-1) \leq \log t + (k-1)$ shares were generated for previous g generations. Therefore, for all $i \in [k-1]$

$$|v_i^{(g)}| \leq \sigma(\log t + (k-1)).$$

Therefore, for $1 < i \leq k-1$

$$|w_{i,j}^{(g)}| \leq \max\{\log t + (k-1), \sigma(\log t + (k-1))\}$$

and for $i = 1$

$$|w_{1,j}^{(g)}| \leq \sigma(\log t + (k-1)).$$

Thus, the total share size in the scheme Π'_k is bounded by:

$$\begin{aligned}
&\log t + (k-1) + \sigma(\log t + (k-1)) + (k-2) \cdot \max\{\log t + (k-1), \sigma(\log t + (k-1))\} \\
&\leq \log t + (k-1) + \sigma(\log t + (k-1)) + (k-2)(\log t + (k-1) + \sigma(\log t + (k-1))) \\
&\leq (k-1)\log t + k \cdot \sigma(\log t + k) + k^2.
\end{aligned}$$

6.3 Proof of Theorem 6.1 assuming Lemma 6.2

Let $k \in \mathbb{N}$ be such that $k \geq 2$. We use the scheme for evolving k-THR constructed in Section 6.1 in which the share size of the t^{th} party is $\sigma_k^{(0)}(t) = kt \cdot \log(kt)$. Using Lemma 6.2 this gives rise to a scheme $\Pi_k^{(1)}$ for evolving k-THR in which the share size of the t^{th} party is:

$$\sigma_k^{(1)}(t) = (k-1) \cdot \log t + k \cdot \sigma_k^{(0)}(\log t + k) + k^2. \tag{6.1}$$

We bound $\sigma_k^{(0)}(\log t + k)$. If $k > \log t$, then

$$\sigma_k^{(0)}(\log t + k) \leq \sigma_k^{(0)}(2k) \leq 2k^2 \log\left(2k^2\right) \leq 4k^2 \cdot \log(2k)$$

If $k \leq \log t$ then

$$\begin{aligned}
\sigma_k^{(0)}(\log t + k) &\leq \sigma_k^{(0)}(2\log t) \\
&\leq k \cdot 2\log t \cdot \log(k \cdot 2\log t) \\
&\leq 2k \cdot \log t \cdot \log\log t + 2k \cdot \log t \cdot \log(2k) \\
&\leq 4k \cdot \log t \cdot \log\log t + 4k^2 \cdot \log(2k),
\end{aligned}$$

where the last inequality follows since $2k \cdot \log t \cdot \log(2k) \leq 2k \cdot \log t \cdot \log\log t + 4k^2 \cdot \log(2k)$. Together we get that

$$\sigma_k^{(0)}(\log t + k) \leq \max\{\sigma_k^{(0)}(2\log t), \sigma_k^{(0)}(2k)\} \leq 4k \cdot \log t \cdot \log\log t + 4k^2 \cdot \log(2k).$$

Plugging this in Equation (6.1), we get that

$$\begin{aligned}\sigma_k^{(1)}(t) &= (k-1) \cdot \log t + k \cdot \sigma_k^{(0)}(\log t + k) + k^2 \\ &\leq (k-1) \cdot \log t + 4k^2 \cdot \log t \cdot \log\log t + 4k^3 \cdot \log(2k) + k^2 \\ &\leq 5k^2 \cdot \log t \cdot \log\log t + 5k^3 \cdot \log k.\end{aligned}$$

Using Lemma 6.2 again, we get a scheme $\Pi_k^{(2)}$ in which the share size of the t^{th} party is

$$\sigma_k^{(2)}(t) = (k-1) \cdot \log t + k \cdot \sigma_k^{(1)}(\log t + k) + k^2. \tag{6.2}$$

We bound $\sigma_k^{(1)}(\log t + k)$ as follows.

$$\begin{aligned}\sigma_k^{(1)}(\log t + k) \leq& \max\{\sigma_k^{(1)}(2\log t), \sigma_k^{(1)}(2k)\} \\ \leq& 5k^2 \cdot \log(2\log t) \cdot \log\log(2\log t) + 5k^2 \cdot \log(2k) \cdot \log\log(2k) + \\ & 5k^3 \cdot \log k \\ \leq& 6k^2 \cdot \log\log t \cdot \log\log\log t + 6k^3 \cdot \log k.\end{aligned}$$

Plugging this back in Equation (6.2), we get that

$$\begin{aligned}\sigma_k^{(2)}(t) &= (k-1) \cdot \log t + k \cdot \sigma_k^{(1)}(\log t + k) + k^2 \\ &\leq (k-1) \cdot \log t + 6k^3 \cdot \log\log t \cdot \log\log\log t + 6k^4 \cdot \log k + k^2 \\ &\leq (k-1) \cdot \log t + 6k^3 \cdot \log\log t \cdot \log\log\log t + 7k^4 \cdot \log k.\end{aligned}$$

Remark. As in the proof of Theorem 5.1, one can iteratively apply Lemma 6.2 again and again to decrease the dependence on $\log\log t \cdot \log\log\log t$. However, the dependence on $\log t$ cannot be improved using this method.

7 A Lower Bound

For general access structures the best standard secret sharing schemes require exponential-size shares. Instantiating our scheme for n parties, results with the n^{th} party holding a share of size 2^{n-1}. Thus, any improvement in the share size on our scheme for general access structures, will imply a non-trivial improvement for general access structures in the standard setting.

In the case of k-threshold access structures, we do not know if our scheme is tight. Specifically, for $k > 2$, using our scheme to implement a standard secret sharing scheme for k-out-of-n is not tight. Indeed, the most significant term in the share size in our scheme depends linearly on $k-1$, while the best schemes in the standard setting are independent of k (see Claim 2.3).

Thus, one may ask whether there exists a secret sharing scheme for the evolving k-THR in which the share size of the t^{th} party is roughly $\log t$. We show that such a scheme cannot exist.

Theorem 7.1 (Theorem 1.3 restated)**.** *For any constant $c \in \mathbb{N}$, there is* no *secret sharing scheme for the evolving* 2-THR *access structure in which the share size of the t^{th} party is at most*

$$\log t + \log\log t + c.$$

Proof. Assume (towards contradiction) that there is a secret sharing scheme for the evolving 2-THR access structure in which the share size of the t^{th} party is at most $\log t + \log\log t + c$ for a constant $c \in \mathbb{N}$. We can use this scheme to implement a standard secret sharing scheme for $(2, n)$-THR in which the share size of party $t \in [n]$ is $m_t \leq \log t + \log\log t + c$.

We use the following claim (reminiscent of Kraft's inequality) that underlies the lower bound of Kilian and Nisan.

Claim 7.2 ([KN90] and [CCX13, Appendix A])**.** For any $n \in \mathbb{N}$ it holds that

$$\sum_{t=1}^{n} \frac{1}{2^{m_t}} \leq 1.$$

Using this claim we get that

$$1 \geq \sum_{t=1}^{n} \frac{1}{2^{m_t}} \geq \sum_{t=1}^{n} \frac{1}{2^{\log t + \log\log t + c}} = \frac{1}{2^c} \cdot \sum_{t=1}^{n} \frac{1}{t \cdot \log t},$$

where we assume that $\log 0 = 0$ and $1/0 = 0$. To get a contradiction we need to show that $\sum_{t=1}^{n} \frac{1}{t \cdot \log t} > 2^c$ for large enough n. Indeed, letting $n \to \infty$, we have that

$$\sum_{t=1}^{\infty} \frac{1}{t \cdot \log t} \geq \int_{1}^{\infty} \frac{1}{t \cdot \log t} dt = \log\log t \to \infty.$$

This completes the proof.

Remark 7.3 (A stronger lower bound)**.** We note that the lower bound can be strengthened to show that even schemes in which the share size is $\sum_{t=0}^{\ell} \log^{(i)}(t) + c$ cannot exist for any $\ell \in \mathbb{N}$ and where $\log^{(i)}(t)$ is the i-times repeated log of t (letting $\log^{(0)}(t) = t$). This follows similarly to the above argument noting that for every $\ell \in \mathbb{N}_0$ it holds that $\int_1^{\infty} \frac{1}{\prod_{i=0}^{\ell} \log^{(i)}(t)} dt = \log^{(\ell+1)} t$ and using that $\log^{(\ell+1)} t \geq 2^c$ for any constant $c \in \mathbb{N}$ and large enough t.

This is reminiscent of bounds in the literature on prefix-free encodings (a.k.a. prefix codes); see [BY76, ER78]. It would be interesting to find structural connections as well.

■

8 Further Work and Open Problems

There are several research directions suggested by this work. The most evident one is to investigate the necessity of the linear dependence on k in the most significant term in our scheme for evolving k-THR. In particular are more algebraic construction possible?

There are several interesting access structures for which we do not have efficient constructions. For example, a very natural evolving access structure is the one in which qualified subsets are

the ones which form a *majority* of the present parties at *some* point in time. The only scheme that realized this access structure we are aware of stems from our construction for general access structures from Section 4 which results with very long shares.

When $k = 2$, the expressions in our upper and lower bounds are identical to the ones appearing in the literature on prefix-free encodings. Is there any relation between prefix-free encodings and secret sharing, e.g. is it possible to construct a prefix free encoding from a evolving 2-THR scheme?

Secret sharing has had many applications in cryptography and distributed computing. One of the most notable examples is *multiparty computation* (MPC). Can secret sharing for evolving access structures be useful for MPC?

We focused on schemes in which correctness and security are *perfect*. One can relax correctness to work with high probability and to allow small statistical error in security. Can this relaxations be used to obtain more interesting and efficient schemes? Another variant of secret sharing schemes is the *computational* one. In these schemes security is required only against computationally bounded adversaries. Efficient computational schemes for much richer classes of access structures are known [Yao, Kra93, VNS+03, KNY14]. Is there a meaningful way to define computationally secure secret sharing schemes for evolving access structures? Can this be used to obtain efficient schemes for more classes of evolving access structures? Cachin [Cac95] studied a similar question in a model in which there is a large public bulletin board.

Other natural variants of secret sharing can be adapted to the evolving setting. For example, verifiable, robust and visual secret sharing. We leave these as interesting directions for future exploration.

References

[Bei11] Amos Beimel. Secret-sharing schemes: A survey. In *Coding and Cryptology - 3rd International Workshop, IWCC*, volume 6639, pages 11–46, 2011.

[BL88] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *8th Annual International Cryptology Conference, CRYPTO*, pages 27–35, 1988.

[Bla79] George R. Blakley. Safeguarding cryptographic keys. *Proceedings of the AFIPS National Computer Conference*, 22:313–317, 1979.

[Bop86] Ravi B. Boppana. Threshold functions and bounded depth monotone circuits. *J. Comput. Syst. Sci.*, 32(2):222–229, 1986.

[BR89] Josh Cohen Benaloh and Steven Rudich. Unpublished. Private Communication with Steven Rudich, 1989.

[BY76] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.*, 5(3):82–87, 1976.

[Cac95] Christian Cachin. On-line secret sharing. In *IMA Conference*, pages 190–198, 1995.

[CCX13] Ignacio Cascudo Pueyo, Ronald Cramer, and Chaoping Xing. Bounds on the threshold gap in secret sharing and its applications. *IEEE Transactions on Information Theory*, 59(9):5600–5612, 2013.

[CT12] László Csirmaz and Gábor Tardos. On-line secret sharing. *Designs, Codes and Cryptography*, 63(1):127–147, 2012.

[Eli75] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

[ER78] Shimon Even and Michael Rodeh. Economical encoding of commas between strings. *Communications of the ACM*, 21(4):315–317, 1978.

[Fri86] Joel Friedman. Constructing $o(n \log n)$ size monotone formulae for the k-th threshold function of n boolean variables. *SIAM J. Comput.*, 15(3):641–654, 1986.

[Geo] National Geographic. Nasa declares end to deep impact comet mission. http://news.nationalgeographic.com/news/2013/09/130920-deep-impact-ends-comet-mission-nasa-jpl/. Accessed: 2016-02-07.

[ISN93] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Multiple assignment scheme for sharing secret. *Journal of Cryptology*, 6(1):15–20, 1993.

[KN90] Joe Kilian and Noam Nisan. Unpublished. See [CCX13], 1990.

[KNR92] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM J. Discrete Math.*, 5(4):596–603, 1992.

[KNY14] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. In *Advances in Cryptology - ASIACRYPT 2014*, pages 254–273, 2014.

[Kra93] Hugo Krawczyk. Secret sharing made short. In *13th Annual International Cryptology Conference, CRYPTO*, pages 136–146, 1993.

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *8th Annual Structure in Complexity Theory Conference*, pages 102–111, 1993.

[PSW13] Rasmus Pagh, Gil Segev, and Udi Wieder. How to approximate a set without knowing its size in advance. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 80–89, 2013.

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

[VNS+03] V. Vinod, Arvind Narayanan, K. Srinathan, C. Pandu Rangan, and Kwangjo Kim. On the power of computational secret sharing. In *4th International Conference on Cryptology in India, INDOCRYPT*, pages 162–176, 2003.

[Wika] Wikipedia. IPv4 address exhaustion. https://en.wikipedia.org/wiki/IPv4_address_exhaustion. Accessed: 2016-02-07.

[Wikb] Wikipedia. Year 2000 problem. https://en.wikipedia.org/wiki/Year_2000_problem. Acccessed: 2016-02-07.

[Yao] Andrew C. Yao. Unpublished. Mentioned in [Bei11]. See also [VNS+03].

A A Simpler Scheme for 3-Evolving-Threshold

In this section we give a slighly simpler construction for a secret sharing scheme for the evolving 3-THR access structure than the construction that follows from Section 6. However, the share size of parties in this scheme are slighly worse for large enough t. One nice property of this construction is the use of the fact that secret sharing several 1-bit secrets via Shamir's scheme can be packed into a single secret sharing scheme with short shares. As before, without loss of generality, we focus on the case where the secret is a single bit.

Theorem A.1. *There is a secret sharing scheme for the evolving* 3-THR *access structure in which the share size of the t^{th} party is bounded by*

$$3\log t + 4\log(\log t + 1) + 3\log(\log(\log t + 1) + 1) + 3.$$

Our main technical claim used to prove Theorem A.1 is given in the following lemma.

Lemma A.2. *Assume that there exists a secret sharing scheme for the evolving* 3-THR *access structure in which the share size of the t^{th} party is $\sigma(t)$. Then, there exists a secret sharing scheme for the evolving* 3-THR *access structure in which the share size of the t^{th} party is*

$$3\log t + \sigma(\log t + 1) + 1.$$

Proof of Theorem A.1 assuming Lemma A.2. The scheme given in Section 4 can be used to obtain a scheme $\Pi^{(0)}$ for evolving 3-THR in which the share size of the t^{th} party is

$$\sigma^{(0)}(t) = 2^{t-1}.$$

Applying Lemma A.2 this gives rise to a scheme $\Pi^{(1)}$ for evolving 3-THR in which the share size of the t^{th} party is

$$\begin{aligned}\sigma^{(1)}(t) &= 3\log t + \sigma^{(0)}(\log t + 1) + 1\\ &\leq 3\log t + t + 1.\end{aligned}$$

Applying Lemma A.2 again we get a scheme $\Pi^{(2)}$ for evolving 3-THR in which the share size of the t^{th} party is

$$\begin{aligned}\sigma^{(2)}(t) &= 3\log t + \sigma^{(1)}(\log t + 1) + 1\\ &\leq 3\log t + 3\log(\log t + 1) + \log t + 1 + 1\\ &= 4\log t + 3\log(\log t + 1) + 2.\end{aligned}$$

Applying Lemma A.2 again we get a scheme $\Pi^{(3)}$ for evolving 3-THR in which the share size of the t^{th} party is

$$\begin{aligned}\sigma^{(3)}(t) =& 3\log t + \sigma^{(2)}(\log t + 1) + 1\\ \leq& 3\log t + 4\log(\log t + 1) + 3\log(\log(\log t + 1) + 1) + 2 + 1\\ \leq& 3\log t + 4\log(\log t + 1) + 3\log(\log(\log t + 1) + 1) + 3.\end{aligned}$$

This proves the theorem. ∎

We proceed with the proof of Lemma A.2.

Proof of Lemma A.2. Let Π be a secret sharing scheme for evolving 3-THR in which the share size of the t^{th} party is $\sigma(t)$. We construct a scheme Π' for the same access structure in which the share size is $3\log t + \sigma(\log t + 1) + 1$.

Let $s \in \{0,1\}$ be the secret to be shared. As in the previous schemes, we assign parties with generations that correspond to the time in which they arrive. The generations are growing in size: For $g = 0, 1, 2 \ldots$ the g^{th} generation begins when the 2^g-th party arrives. Therefore, the number of parties that are part of generation g is $\text{SIZE}(g) = 2^g$ and party $t \in \mathbb{N}$ is part of generation $g = \lfloor \log t \rfloor$.

Whenever a generation begins the dealer prepares shares for all parties that are part of that generation. Let us focus on the beginning of the g^{th} generation and describe the dealer's procedure:

1. Split s using a secret sharing scheme for $(3, \text{SIZE}(g))$-THR. Denote the resulting shares by $u_1^{(g)}, \ldots, u_{\text{SIZE}(g)}^{(g)}$.
2. Generate one share using the evolving scheme Π given the secret s and previous shares $\{v^{(i)}\}_{i \in \{0,\ldots,g-1\}}$. Denote the resulting share by $v^{(g)}$.
3. Sample two random bits $b_1^{(g)}, b_2^{(g)} \in \{0,1\}$.
4. Split the length $g+1$ string $b_2^{(g)} || s \oplus b_1^{(0)} || \ldots || s \oplus b_1^{(g-1)}$ using $(2, \text{SIZE}(g))$-THR. Denote the resulting shares by $w_1^{(g)}, \ldots, w_{\text{SIZE}(g)}^{(g)}$.
5. Set the secret share of the j^{th} party in the g^{th} generation (i.e., $j \in [\text{SIZE}(g)]$) to be
$$\left(u_j^{(g)}, v^{(g)}, b_1^{(g)}, w_j^{(g)}, \{s \oplus b_2^{(i)}\}_{i \in \{0,\ldots,g-1\}}\right).$$

Correctness and security. Let $t_1, t_2, t_3 \in \mathbb{N}$ be any two different parties. We show that the secret s can be computed from their shares. There are four cases to consider (without loss of generality) depending on the generations of t_1, t_2 and t_3 that we denote by g_1, g_2 and g_3 respectively: (1) $g_1 = g_2 = g_3$, (2) $g_1 = g_2 < g_3$, (3) $g_1 < g_2 = g_3$, and (4) $g_1 < g_2 < g_3$. In case (1) the parties can use the $u^{(g_1)}$ part of the share to execute the reconstruction procedure of the $(3, \text{SIZE}(g_1))$-THR scheme. In case (2) t_1 and t_2 can use their $w^{(g)}$ share to learn $b_2^{(g)}$ and XOR with the share $s \oplus b_2^{(g_1)}$ that t_3 holds. In case (3) t_2 and t_3 can use their $w^{(g)}$ shares to compute $s \oplus b_1^{(g_1)}$ and then XOR with the bit $b_1^{(g_1)}$ that t_1 posses. Finally, in case (4) the parties posses three shares $v^{(g_1)}$, $v^{(g_2)}$ and $v^{(g_3)}$ for the evolving 3-THR scheme Π that can be used to learn s.

For security we consider any two parties t_1 and t_2 from generations g_1 and g_2, respectively, and the following two cases (without loss of generality): (1) $g_1 = g_2$, and (2) $g_1 < g_2$. In both cases one can verify that both players do not have enough information to learn anything about the secret s.

Share size analysis. Denote by $\sigma(t)$ the share size of party t in the scheme Π. We bound the size of each component in the share of party t. The share of party t that is the j^{th} party of generation $g = \lfloor \log t \rfloor$ is $(u_j^{(g)}, v^{(g)})$.

1. $u_j^{(g)}$ – generated by secret sharing s using a scheme for $(3, \text{SIZE}(g))$-THR. Since $\text{SIZE}(g) = 2^g$ and using Claim 2.3 we get that
$$|u_j^{(g)}| \leq \log\left(\text{SIZE}(g)\right) \leq \lfloor \log t \rfloor.$$

2. $v^{(g)}$ – generated by generating one share of a secret sharing scheme Π for evolving 2-THR. Recall that g shares were generated for previous generations. Therefore,

$$|v^{(g)}| = \sigma(g+1) = \sigma(\lfloor \log t \rfloor + 1).$$

3. $b_1^{(g)}$ – a single bit.

4. $w_j^{(g)}$ – generated by secret sharing a string of length $g+1$ using $(2, \text{SIZE}(g))$-THR. Since $\text{SIZE}(g) = 2^g$ and using Claim 2.3 we get that

$$|w_j^{(g)}| \leq \log(\text{SIZE}(g)) \leq \lfloor \log t \rfloor.$$

5. $\{s \oplus b_2^{(i)}\}_{i \in \{0,\ldots,g-1\}}$ – $g = \lfloor \log t \rfloor$ bits.

Thus, the total share size in the scheme Π' is bounded by

$$3 \log t + \sigma(\log t + 1) + 1.$$

■

ECCC ISSN 1433-8092

http://eccc.hpi-web.de