Evolving Secret Sharing Made Short

Danilo Francati[1] and Daniele Venturi[2]

[1]Aarhus University
[2]Sapienza University of Rome

October 7, 2023

Abstract

Evolving secret sharing (Komargodski, Naor, and Yogev, TCC'16) generalizes the notion of secret sharing to the setting of *evolving access structures*, in which the share holders are added to the system in an online manner, and where the dealer does not know neither the access structure nor the maximum number of parties in advance. Here, the main difficulty is to distribute shares to the new players without updating the shares of old players; moreover, one would like to minimize the share size as a function of the number of players.

In this paper, we initiate a systematic study of evolving secret sharing in the *computational setting*, where the maximum number of parties is polynomial in the security parameter, but the dealer still does not know this value, neither it knows the access structure in advance. Moreover, the privacy guarantee only holds against computationally bounded adversaries corrupting an unauthorized subset of the players.

Our main result is that for many interesting, and practically relevant, evolving access structures (including graphs access structures, DNF and CNF formulas access structures, monotone circuits access structures, and threshold access structures), under standard hardness assumptions, there exist efficient secret sharing schemes with computational privacy and in which the shares are *succinct* (i.e., much smaller compared to the size of a natural computational representation of the evolving access structure).

Keywords: secret sharing, evolving access structures, computational security.

Contents

1 Introduction

A (threshold) secret sharing scheme, as introduced independently by Blakley [10] and Shamir [37], allows a dealer to share a secret message μ between n parties, obtaining shares $\sigma_1, \ldots, \sigma_n$, in such a way that the following two properties are satisfied for a fixed threshold parameter $t \leq n$:

- **Correctness:** Any subset of at least t parties can reconstruct the message (by pulling their shares together).
- **(Perfect) Privacy:** Any subset of at most $t-1$ parties obtains no information (statistically) on the message.

Ito, Saito, and Nishizeki [21] extended the concept of secret sharing to general *access structures* $\mathcal{A}$, in which certain subsets of the n parties are allowed to reconstruct the message (i.e., the so-called *authorized* subsets $\mathcal{I} \in \mathcal{A}$), and all other subsets of the n parties (i.e., the so-called *unauthorized* subsets $\mathcal{U} \notin \mathcal{A}$) obtain no information on the message. It is natural to require that access structures should be *monotone*, meaning that if a subset $\mathcal{I}$ is authorized, then any subset of the parties that includes $\mathcal{I}$ is also authorized.

An important efficiency measure in the context of secret sharing schemes is the *share size*, defined as the bit-length of the largest share given to the n parties. Indeed, a large body of work has focused on minimizing the share size for different access structures $\mathcal{A}$. For instance, Shamir's scheme (based on polynomial interpolation) for threshold access structures achieves share size $\max\{\ell, \log n\}$, where ℓ is the length of the message, which is known to be optimal [34]. More generally, Csirmaz [12, 13] proved that there is an (explicit) access structure that requires a *total* share size of $\Omega(n^2/\log n)$. In contrast, the best scheme [2] for general access structures achieves share size $1.5^{n+o(n)}$, which is pretty far from the lower bound. We refer to the excellent survey by Beimel [6] for an overview of the main results in classical secret sharing.

Evolving secret sharing. In a beautiful work, Komargodski, Naor, and Yogev [23], generalized secret sharing to the setting of *evolving* access structures, in which the number of parties n is not known in advance and can potentially go to infinity. Intuitively, an evolving access structures consists of a monotone sequence of subsets $\mathcal{A} = \{\mathcal{A}_n\}_{n\geq 1}$, where $\mathcal{A}_n$ denotes the access structure when there are n parties. Importantly, the dealer does not know n, neither it knows the access structure $\mathcal{A}_n$ before the n-th party enters the system. The main result in [23] is a secret sharing scheme for the evolving threshold access structure (i.e., the number of parties n grows, but the threshold $t = \mathsf{poly}(\lambda)$ is fixed and known to the dealer), in which the share size of party n, for messages of size $\ell = 1$, is $(t-1)\cdot \log n + O(\log\log n)$. They also give a secret sharing scheme for any evolving access structure, in which the share size of party n is 2^{n-1} (i.e., with exponential share size).

In a follow-up work, Komargodski and Paskin-Cherniavsky [25] give a secret sharing scheme for the more general evolving *dynamic* threshold access structure, which is represented by a sequence of thresholds $\{t_n\}_{n\geq 1}$ of increasing[1] size, so that the authorized subsets when there are n parties are all the subsets of at least t_n parties. The share size of this construction, when the message length is $\ell = 1$, is $O(n^4 \cdot \log n)$. Note that, in this case, each of the thresholds t_n can depend on n; this captures, e.g., the so-called *evolving majority* access structure, in which qualified subsets are those which form a majority of the current number of parties at *some* point in time. Xin and Yuan [40] show that the share size can be improved to $O(n^4)$ by relying on techniques from algebraic geometry.

[1] The fact that the thresholds should be increasing is required to ensure monotonicity of the access structure.

More recently, at Eurocrypt'20, Beimel and Othman [9] gave constructions of secret sharing schemes for the dynamic threshold *ramp* access structure, which is represented by two functions $\tau, \rho : \mathbb{N} \to \mathbb{N}$ such that, when there are n parties in the system, the threshold for reconstruction is $\rho(n)$ (i.e., any subset of at least $\rho(n) \leq n$ parties can reconstruct the message) while the threshold for privacy is $\tau(n)$ (i.e., every subset of at most $\tau(n) < \rho(n)$ parties has no information about the message). The share size in their construction depends on the gap between the reconstruction and privacy thresholds; in particular, when the message length is $\ell = 1$, the share size is $\mathsf{polylog}(n)$ when $\rho(n) - \tau(n) = n/\mathsf{polylog}(n)$, and $O(n \cdot \log n)$ when $\rho(n) - \tau(n) = \sqrt{n}$; furthermore, they also give a direct construction for the special case in which $\rho(n) = t$ and $\tau(n) = t/2$ for any constant $t = O(1)$, with share size $O(\log t \cdot \log n)$. This improves over a previous construction by the same authors [8], which achieves share size $O(1)$, always when the message length is $\ell = 1$, but when $\rho(n) = b \cdot n$ and $\tau(n) = a \cdot n$ for any constants $0 < a < b < 1$ (i.e., when the gap between the reconstruction and the privacy thresholds is a constant fraction of the number of parties).

All of the above constructions achieve *perfect* privacy (i.e., unauthorized subsets have no information about the message, in an information-theoretic sense), and can potentially accommodate an infinite number of users. Moreover, they are all tailored to variations of the evolving threshold access structure, and have share size that is at least linear in the number of parties n or in the threshold t (the only exception being ramp secret sharing schemes). Given this state of affairs, the following question arises naturally:

> *Can we get secret sharing schemes with* succinct *shares (e.g., with size independent on n) for richer* evolving *access structures, possibly under computational assumptions?*

1.1 Our Contributions

We provide a positive answer to the above question by initiating a systematic study of secret sharing schemes for evolving access structures in the *computational setting*. Our contributions are summarized in Table 1, where we compare our results with the state of the art in terms of access structure, share size, and computational assumptions.

In a nutshell, we provide constructions of *computationally private* secret sharing schemes for a plethora of evolving access structures, under standard hardness assumptions. In all of our constructions, the number of parties n is upper bounded by an arbitrary polynomial in the security parameter, but the dealer does not know this polynomial, neither it knows the overall access structure (it only knows the new authorized subsets when a new party joins the system). For some access structures, the share size is *succinct* (i.e., much smaller compared to the size of a natural computational representation of the evolving access structure). More in details, we given constructions of secret sharing schemes:

- For any evolving access structure in which the n-th participant appears in at most $d_n = \mathsf{poly}(\lambda, n)$ authorized subsets. This construction requires one-way functions (OWFs), and yields share size $\lambda \cdot (d_n + 1)$.
- For the dynamic threshold access structure. This construction requires OWFs, and yields share size $\lambda \cdot (n + 1)$
- For graphs access structures, in which the parties are added to the nodes of an evolving (undirected) graph, and the authorized subsets consist of all the pair of nodes for which there is an edge in the graph. This construction requires either the RSA assumption,

Reference	Access Structure	Parameters	Share Size	Assumptions
[23]	Any	–	$\lambda \cdot 2^{n-1}$	–
	Static Threshold	$t = \mathsf{poly}(\lambda)$	$(t-1) \cdot \log n + \mathsf{poly}(\lambda, t) \cdot o(\log n)$	–
[25]	Dynamic Threshold	$t_1 \leq \cdots \leq t_n \leq n$	$\lambda \cdot O(n^4 \cdot \log n)$	–
[8]	Ramp Dynamic Threshold	$\rho(n) - \tau(n) = c \cdot n$	$O(\lambda)$	–
[9]	Ramp Dynamic Threshold	$\rho(n) - \tau(n) = n/\mathsf{polylog}(n)$	$\lambda \cdot \mathsf{polylog}(n)$	–
	Ramp Dynamic Threshold	$\rho(n) - \tau(n) = \sqrt{n}$	$\lambda \cdot O(n \cdot \log n)$	–
	Ramp Static Threshold	$\rho(n) = t; \tau(n) = t/2$	$\lambda \cdot O(\log t \cdot \log n)$	–
[40]	Static Threshold	$0 < t \leq n; \epsilon > 0$	$\lambda \cdot O(t^{1+\epsilon} \cdot \log n)$	–
	Dynamic Threshold	$t_1 \leq \cdots \leq t_n \leq n$	$\lambda \cdot O(n^4)$	–
§4.2	Any	$d_n = \mathsf{poly}(\lambda, n)$	$\lambda \cdot (d_n + 1)$	OWFs
§5.1	Static Threshold	$t = \mathsf{poly}(\lambda)$	λ	–
	Dynamic Threshold	$t_1 \leq \cdots \leq t_n \leq n$	$\lambda \cdot (n+1)$	OWFs
§5.2	Graphs	–	$\mathsf{poly}(\lambda)$	RSA/iO + SSB
§5.3	Monotone Circuits	$m_g = m_g^{\wedge} + m_g^{\vee}; g \geq 1$	$m_g \cdot \mathsf{poly}(\lambda)$ $(m_g^{\vee})^2 + m_g^{\wedge} \cdot O(\lambda)$ $m_g^{\vee} + m_g^{\wedge} \cdot O(\lambda)$	RSA/iO + SSB DDH/BDDH LWE
§5.4	CNF Formulas	$m \geq 1$	$\mathsf{poly}(\lambda)$ $m^2 \cdot \mathsf{poly}(\lambda)$ $m \cdot \mathsf{poly}(\lambda)$	RSA/iO + SSB DDH/BDDH LWE
§5.5	DNF Formulas	$t_n \geq 1$	$\lambda \cdot (t_n + 1)$	OWFs

Table 1: Comparing our results with state-of-the-art constructions for evolving secret sharing, in terms of access structure, share size, and computational assumptions. See Section 1.2 for the definition of the various parameters. The share size refers to the size of the share received by the n-th party, where n is the current number of parties in the system. In information-theoretic constructions, n is unbounded; in the computational setting $n = \mathsf{poly}(\lambda)$ (but the dealer does not know an upper bound on n). For simplicity, we only consider message length $\ell(\lambda) = \lambda$; when a scheme is for $\ell(\lambda) = 1$, we consider the share size obtained by repeating the sharing procedure λ times in parallel.

or indistinguishability obfuscation (iO) and somewhere statistically binding (SSB) hash functions, and yields share size $\mathsf{poly}(\lambda)$.

- For monotone circuits access structures, in which the parties correspond to the inputs wires of an evolving boolean circuit made of AND and OR gates with unbounded fan-in. This construction requires either of the following assumptions: (i) RSA, (ii) iO plus SSB hash functions, (iii) DDH/BDDH, (iv) LWE, and yields share size that is roughly linear[2] in the number of gates that are added to the circuit.

- For CNF and DNF formulas access structures, which are a special case of monotone circuits. In fact, in these cases, we give direct constructions that are slightly better in terms of assumptions and/or share size.

All of the above constructions allow to share secret messages of size $\ell(\lambda) = \lambda$. In the final part of the paper, we deal with the problem of domain extension for evolving secret sharing schemes and show that, under mild assumptions, all of our schemes can be generically upgraded to support messages of length $\ell(\lambda) = \mathsf{poly}(\lambda)$, by paying only an additive (in fact, linear in ℓ) overhead in terms of share size. This transformation only requires OWFs.

[2]This is a very rough approximation. See Table 1 and Section 1.2 for more precise parameters.

1.2 Technical Overview

We now give a detailed overview of the main techniques we use in order to obtain our results, starting with the notion of computational privacy for evolving secret sharing, and then explaining the ideas behind each of our constructions.

Computational evolving secret sharing. The definition of computationally private secret sharing for evolving access structures is the natural adaptation of the corresponding information-theoretic definition. In particular, in the computational setting, the number of parties is $n = \mathsf{poly}(\lambda)$. An evolving access structure is a monotone sequence $\{\mathcal{A}_n\}_{n\geq 1}$, where $\mathcal{A}_n$ denotes the access structure when there are n parties in the system. We note that the dealer does not know the actual polynomial that upper bounds n, neither it knows the access structure $\mathcal{A}_n$ before the n-th party enters the system. This limitation is rather important, as if the dealer knows that $n < \tilde{n}$ for some polynomial $\tilde{n}$, along with the access structure $\mathcal{A}_{\tilde{n}}$, it can simply use a standard secret sharing scheme for $\mathcal{A}_{\tilde{n}}$ and distribute the shares to the parties as they arrive.

Computational privacy simply requires that for every (polynomial) $n \geq 1$, for all unauthorized subsets $\mathcal{U} \notin \mathcal{A}_n$, and for every pair of messages (μ_0, μ_1), no computationally bounded adversary given the shares $(\sigma_i)_{i\in\mathcal{U}}$ can distinguish, with better than negligible probability, whether the shares are generated using message μ_0 or message μ_1.

Rigidity. In principle, an evolving access structure $\mathcal{A} = \{\mathcal{A}_n\}_{n\geq 1}$ only needs to be monotone, in the sense that, for every $\mathcal{I}$ such that $\mathcal{I} \in \mathcal{A}_n$, every subset $\mathcal{I}' \subset \mathcal{I}$ also satisfies $\mathcal{I}' \in \mathcal{I}$. Now, say that $n = 10$ and that parties 1 and 3 are not authorized (i.e., $\mathcal{U} = \{1, 3\} \notin \mathcal{A}_{10}$); when party 11 arrives, the set $\mathcal{U}$ might become authorized without violating monotonicity. An access structure is called *rigid* if the above never happens, namely the subset $\mathcal{U}$ never becomes authorized.

Interestingly, the definition of evolving access structures adopted in all previous works automatically implies rigidity; indeed, Komargodski *et al.* [23] (and all the follow-up papers) define $\mathcal{A}_n$ as the intersection of the entire access structure $\mathcal{A}$ (for $n \to \infty$) with $[n]$. This way, it does not matter *when* a subset becomes authorized: if the subset $\{1, 3\}$ becomes authorized when $n = 100$, we still have that $\{1, 3\} \in \mathcal{A}_3 = \mathcal{A} \cap [3]$ by definition (which excludes the above example). Note that some access structures, as the evolving dynamic threshold access structure, are also automatically rigid.

It turns out that rigidity plays a rather important role when defining how more complex access structures (e.g., monotone circuits) can evolve. Indeed, in Section 3.2, we show that in any secret sharing for a *non-rigid* evolving access structure $\mathcal{A}$, the dealer must update the shares of old players at some point. This observation becomes immediately apparent when we look again at the above example: Since $\mathcal{U} = \{1, 3\}$ is unauthorized when $n = 10$, by the privacy property, the shares σ_1 and σ_3 reveal no information about the message; hence, unless we update these shares, σ_1 and σ_3 are not enough to reconstruct the message when $n = 11$, which contradicts the correctness property.

General access structures. In Section 4, we give a simple construction of a secret sharing scheme for general rigid evolving access structures $\hat{\mathcal{A}}$. However, the scheme is provably secure (and efficient) only when we make the restriction that each party n is added to $d_n = \mathsf{poly}(\lambda, n)$ authorized subsets. The share size of party n is going to be $\lambda \cdot (d_n + 1) = \mathsf{poly}(\lambda, n)$. We remark that some kind of limitation on the access structure $\hat{\mathcal{A}}$ is inherent, as Mazor [30] proves that there is a (rigid) evolving access structure $\hat{\mathcal{A}}$ such that every secret sharing scheme for $\hat{\mathcal{A}}$ has shares of size $2^{n-o(n)}$ for infinitely many n's.

This construction roughly works as follows. Let G be a pseudorandom generator (PRG) with unbounded polynomial stretch (this exists assuming OWFs); we think of the PRG output as a sequence of λ-bit blocks, which we denote by $\mathsf{G}(\kappa)[j]$. The share σ_n given to the n-th party consists of a λ-bit seed κ_n for the PRG, as well as a ciphertext $\gamma_{\mathcal{I}}$ for every new subset $\mathcal{I}$ containing n that is added to the access structure $\hat{\mathcal{A}}_n$. This ciphertext is obtained by masking the message μ with a pad $\rho_{\mathcal{I}} = \bigoplus_{i \in \mathcal{I}} \mathsf{G}(\kappa_i)[j(\mathcal{I})]$, where κ_i is the seed of party i, and $j(\mathcal{I})$ is the index corresponding to the subset $\mathcal{I}$ in some lexicographic order. Note that the number of ciphertexts given to party n is exactly d_n; furthermore, since both n and d_n are polynomial, the number of PRG blocks that are needed is still polynomial, and thus the construction is efficient.

Correctness follows by observing that the parties corresponding to an authorized subset $\mathcal{I} \in \hat{\mathcal{A}}_n$ can recover the pad $\rho_{\mathcal{I}}$ and reveal the message. Privacy follows by an hybrid argument, in which we replace the ciphertext $\gamma_{\mathcal{I}}$ for every $\mathcal{I}$ that contains an index corresponding to an honest party with a random string. Since the attacker does not know the seed of honest players, these hybrids are all computationally indistinguishable by security of the PRG. Hence, one observes that in the final hybrid, the distribution of the shares corresponding to an unauthorized subset of the players is independent of the message. Interestingly, our construction shares similarities with an old scheme proposed by Cachin in the context of *online* secret sharing [11, 14]. The main difference is that Cachin's scheme directly uses OWFs (or hard-core bits) instead of PRGs, and additionally requires a public bulletin board that must be updated by the dealer.[3] However, in retrospect, the only reason why the bulletin board is needed is because Cachin's definition of evolving access structure does not require rigidity. This is consistent with our impossibility result showing that secret sharing schemes for non-rigid evolving access structures require the dealer to update old shares.

Dynamic threshold access structures. Next, we move to concrete access structures that do not obey the restrictions of the above generic construction, starting with the evolving threshold access structure. The static case, where there is a single threshold $t = \mathsf{poly}(\lambda)$ known to the dealer, is rather simple to deal with: although the dealer does not know the maximum number of users, it knows that $n = \mathsf{poly}(\lambda)$ and thus can define the share of the n-th party as the evaluation $\sigma_n = f(n)$ of a random polynomial f of degree $t-1$ with coefficients over a field of exponential size 2^λ, subject to the constraint that $f(0) = \mu$; this yields share size λ, ensures both correctness and privacy as per Shamir's secret sharing, and moreover allows to accommodate an arbitrary polynomial number of users.[4]

Hence, we move to the more challenging setting of the evolving dynamic threshold access structure. As explained above, this access structure is specified by a sequence of thresholds $\{t_n\}_{n \geq 1}$ such that $t_n \geq t_{n-1}$, and the authorized subsets when there are n parties are all the subsets of size at least $0 < t_n \leq n$. Here, the threshold t_n may depend on n, and the dealer does not know t_n before party n arrives. Our construction, which can be found in Section 5.1, works as follows. Let G be a PRG with unbounded polynomial stretch. When party n arrives, the dealer samples a random seed κ_n, along with a random polynomial f_n of degree $t_n - 1$ over a field of exponential size 2^λ, subject to the constraint that $f_n(0) = \mu$. The share σ_n of party n consists of the seed κ_n, of the evaluation $f_n(n)$, along with $n-1$ ciphertexts $(\gamma_i)_{i<n}$, where each

[3] Online secret sharing is the ancestor of evolving secret sharing. Csirmaz and Tardos [14] show how to remove the public bulletin board, obtaining information-theoretic privacy for infinitely many parties; however, they require the dealer knows an upper bound on the maximum number of authorized subsets a party can join.

[4] Note that, in case $t = O(1)$, the evolving threshold access structure actually satisfies the condition required by our generic construction, as the number of new authorized subsets in which party n participates is exactly $d_n = \binom{n}{t} = \mathsf{poly}(\lambda)$. However, the above direct construction based on Shamir secret sharing works even for $t = \mathsf{poly}(\lambda)$, and does not require computational assumptions.

ciphertext γ_i is an encryption of $f_n(i)$ under the pad $\rho_i = \mathsf{G}(\kappa_i)[n-i]$ (i.e., the next unused block output by the PRG $\mathsf{G}(\kappa_i)$ associated to party i).

Correctness follows by observing that the parties corresponding to an authorized subset can recover at least t_n pads used to encrypt the evaluations of the polynomial f_n, and thus reconstruct the message via polynomial interpolation. Privacy follows by an hybrid argument, in which we replace all of the ciphertexts γ_j corresponding to honest parties with a uniformly random ciphertext. Since the attacker does not know the seed of honest players, these hybrids are all computationally indistinguishable by security of the PRG. Hence, one observes that for every $k \in [n]$, the adversary now knows at most $t_k - 1$ evaluations of the polynomial f_k, and thus the message is information-theoretically hidden.

An interesting feature of our construction is that it works unmodified even assuming the sequence of thresholds $\{t_n\}_{n\geq 1}$ is not increasing. Namely, it is allowed that $t_n < t_{n-1}$ for some number of parties n, so long as the newly added authorized subsets (i.e., those that are in $\hat{\mathcal{A}}_n$, but not in $\hat{\mathcal{A}}_{n-1}$) always include party n. The latter ensures monotonicity (and rigidity is also preserved). We find this to be a natural extension of the evolving dynamic threshold access structure.

Graphs access structures. Consider now the case of graphs access structures, in which the parties correspond to nodes $v_1, \ldots, v_n$ in an undirected graph $G = (\mathcal{V}, \mathcal{E})$, and parties (i, j) are authorized if and only if $(v_i, v_j) \in \mathcal{E}$ (i.e., (v_i, v_j) is an edge in the graph). Applebaum *et al.* [1] recently gave a *succinct* secret sharing scheme for this access structure, based on a new primitive called *projective* PRG. Intuitively, a projective PRG, as any standard PRG, allows to stretch a short seed msk (a.k.a. the master secret key) into a pseudorandom string of *bounded* length $m \in \mathbb{N}$. Additionally, one can use the master secret key msk to generate projective keys $\alpha_{\mathcal{T}}$ associated to sets $\mathcal{T} \subset [m]$, in such a way that it is possible to use $\alpha_{\mathcal{T}}$, along with the master public key mpk associated to msk, to recover the PRG output corresponding to the seed msk in the positions indexed by $\mathcal{T}$. On the other hand, all the remaining positions still look pseudorandom (this property is known as *robustness*). For this notion to be non-trivial, the projective keys must be succinct (i.e., with length sub-linear in $|\mathcal{T}|$).

Applebaum *et al.* [1] first give a construction for *bipartite* graphs access structures, in which the nodes belong to two sets $\mathcal{V} = (\mathcal{V}^{(0)}, \mathcal{V}^{(1)})$, and the edges are only between nodes pertaining to different sets. It is known that secret sharing schemes for bipartite graphs imply ones for arbitrary graphs. To secret share a message $\mu \in \{0, 1\}$, the dealer picks a random seed msk for the projective PRG, and lets $y_1, \ldots, y_m$ be the full PRG output. Hence, the share of a left node $v_i \in \mathcal{V}^{(0)}$ is set to $\mu \oplus y_i$, whereas the share of a right node $v_j \in \mathcal{V}^{(1)}$ is set to the projective key for the set $\mathcal{T}_j = \{i : (v_i, v_j) \in \mathcal{E}\}$ of v_j's neighbors.

In Section 5.2, we extend the above construction to the evolving setting. Here, the underlying graph evolves as nodes and edges are added to it, whenever new players enter the system. However, new edges can be added only if those involve a new player, otherwise we would violate rigidity. This yields the rigid evolving bipartite graphs access structure. Our construction is based on the following observation: when the n-th party corresponds to a right node $v_n \in \mathcal{V}^{(1)}$, the dealer can define its share as the projective key corresponding to the set of left neighbors $\mathcal{T}_n$ of node v_n. On the other hand, when the n-th party corresponds to a left node $v_n \in \mathcal{V}^{(0)}$, we would need to encrypt the message with the next available output bit from the projective PRG. This generates two technical problems:

- The output length of the projective PRG is fixed to some value m, which needs to be known in advance. We solve this problem by requiring that the projective PRG has *unbounded* polynomial stretch, so that the setup does not depend on m. Note that increasing the stretch of a projective PRG (in a black-box way) is a non-trivial task: For instance, the

standard trick to increase the stretch of a PRG by running it sequentially poly-many times simply does not work, as it destroys succinctness. Fortunately, in Section 2.8.3, we show that some of the constructions in Applebaum *et al.* [1] can be adapted to our purpose. In particular, there exist projective PRGs with unbounded polynomial stretch and, with master public keys and projective keys of size $\mathsf{poly}(\lambda)$, assuming either the RSA assumption, or indistinguishability obfuscation (iO) and somewhere statistically binding (SSB) hash functions.

- Setting the share σ_n to $\gamma_n = \mu \oplus y_n$, where y_n is the next output bit produced by the projective PRG, requires to update the shares of v_n's right neighbors, as the projective keys given to these nodes needs to allow them to recover y_n. We solve this problem by repeating the construction in parallel two times, namely the share of party n is obtained by considering two independent executions of the construction by Applebaum *et al.* [1]: one in which v_n is a left node, and one in which v_n is a right node. This way, assuming rigidity, when a new party enters the system, we never need to update old shares.

The above yields a secret sharing scheme for the rigid evolving *bipartite* graphs access structure, with shares of size $\mathsf{poly}(\lambda)$, under either the RSA assumption or assuming iO and SSB hash functions. Additionally, the latter directly implies an evolving secret sharing scheme, with the same share size, for (non-bipartite) graphs access structures.

Monotone circuits access structures. Finally, consider the case of monotone circuits access structures, in which the parties correspond to an input $x = (x_1, \ldots, x_n)$ for a boolean circuit C consisting of AND and OR gates with unbounded fan-in, and a subset of the parties $\mathcal{I}$ is authorized if $C(x_{\mathcal{I}}) = 1$, where $x_{\mathcal{I}}$ is the input associated to $\mathcal{I}$ (i.e., $x_i = 1$ if $i \in \mathcal{I}$ and $x_i = 0$ otherwise). For simplicity, let us assume[5] that the circuit alternates layers made only by either OR or AND gates, starting always with OR gates; hereafter, we refer to such circuits as AND-OR circuits. Applebaum *et al.* [1] give a construction of a *succinct* secret sharing scheme for this access structure, based on projective PRGs. In particular, the share size in their construction grows with the number of gates in the circuit, which improves over the classical construction of Yao [39], in which the share size grows linearly with the number of wires in the circuit.

At a high level, the construction by Applebaum *et al.* [1] allocates to the i-th gate a secret key κ_i, and makes sure that a set of authorized parties corresponding to input $x \in \{0,1\}^n$ will be able to learn the keys of the gates that are satisfied by x, while all other keys remain secret. The keys associated to the OR gates are pseudorandom blocks from the output of the projective PRG, whereas the keys associated to the AND gates are the projective keys for the set $\mathcal{T}_i = \{j : i \to j \text{ in } C\}$ corresponding to the out-neighbors of the i-th gate. The share of the parties include the master public key mpk of the projective PRG, as well as a ciphertext for each AND gate, which essentially allows one to move from an OR gate to an AND gate during reconstruction; the only exception are the input OR gates, for which the secret key is a random label that is given in the clear to the associated players.

In Section 5.3, we extend the above construction to the evolving setting. Here, the underlying circuit evolves as wires and gates are added to it, whenever new players enter the system. Some care is needed when specifying how the circuit evolves: say that we have a circuit over $n-1$ inputs; when the n-th player arrives, monotonicity may forbid to add the corresponding input wire as input to an already existing AND gate (e.g., if the AND gate is an output gate). Still, we could add such wires as part of other AND and OR gates, or add new gates to the circuit.

[5] This assumption is essentially without loss of generality, at least if one is willing to pay an additive factor of n in the number of OR gates, which does not impact the final share size by too much. See Section 5.3 for more on this point.

However, the latter cannot be done arbitrarily if we want to also consider rigidity, which, as explained above, is a necessary condition in order to have an evolving secret sharing scheme where old shares do not need to be updated. To allow more flexibility, we will consider a strict generalization in which the parties arrive in generations instead of one by one. We denote by $n_g \geq 1$ the number of parties in generation $g \geq 1$, so that $n = \sum_g n_g$. Hence, when the first generation arrives, the access structure is specified by a circuit $\hat{C}_1(x_1, \ldots, x_{n_1})$ for some AND-OR circuit $\hat{C}_1$; when the second generation arrives the access structure is specified by the circuit

$$\hat{C}_1(x_1, \ldots, x_{n_1}) \vee \hat{C}_2(x_1, \ldots, x_{n_1+n_2}),$$

where $\hat{C}_2$ is any AND-OR circuit such that $\hat{C}_2(x_1, \ldots, x_{n_1}, 0, \ldots, 0) = 0$. The above preserves monotonicity (as the output of the two circuits are input to an OR gate), and ensures rigidity (as any assignment $x_1, \ldots, x_{n_1}$ that does not satisfy $\hat{C}_1$ won't satisfy $\hat{C}_2$ unless some of the parties in $[n] \setminus [n_1]$ is present).[6]

Given the above characterization, our construction proceeds as follows. When the n-th generation begins, the dealer knows the circuit $\hat{C}_g$; say such a circuit is made of $m_g = m_g^{\vee} + m_g^{\wedge}$ gates, where $m_g^{\vee}$ (resp., $m_g^{\wedge}$) denotes the number of OR (resp., AND) gates. The dealer now distributes the shares to the parties of the g-th generation as in the construction by Applebaum *et al.* [1], with the only difference that the secret key associated to the input OR gates is defined using a standard PRG with unbounded polynomial stretch, evaluated using a seed that is only known by the corresponding player. This way, the wires associated to old players can be used as inputs in the new circuits, without the need for updating old shares. Moreover, the dealer knows the sub-circuit representing each generation, and thus can define the secret keys associated to the OR gates using a fresh pair of keys $(\mathsf{mpk}_g, \mathsf{msk}_g)$ for a projective PRG with *bounded* output length $m_g^{\vee}$. By using the constructions of projective PRGs from Applebaum *et al.* [1], we can instantiate our scheme from either (i) the RSA assumption, or (ii) iO and SSB hash functions, or (iii) the DDH/BDDH assumption, or (iv) the LWE assumption, with different trade-offs in terms of share size (see Table 1).

We remark that the rigid evolving monotone circuits access structure captures several interesting evolving access structures as a special case, such as:

- The rigid evolving monotone conjunctive normal form (CNF) formulas access structure, in which the authorized subsets $\mathcal{I}$ are those corresponding to inputs $x_{\mathcal{I}}$ such that $\bigwedge_{i \in [m]} C_i$, where each clause C_i is a disjunction over a subset of the n players. Note that, while n increases, the clauses change over time. However, by monotonicity, no clause can be added to the access structure, and, by rigidity, no clause can be removed from the access structure. Thus, the number of clauses m is a fixed parameter of the access structure.

- The rigid evolving monotone disjunctive normal form (DNF) formulas access structure, in which the authorized subsets $\mathcal{I}$ are those corresponding to inputs $x_{\mathcal{I}}$ such that $\bigvee_{i \in [m]} C_i$, where each clause C_i is a conjunction over a subset of the n players. Note that, while n increases, the clauses change over time. However, by monotonicity, we can neither remove old clauses nor add new variables to old clauses, and, by rigidity, we can add new clauses, so long as each new clause including old inputs must also include at least one new input. Thus, the number of clauses m is not fixed, and grows over time together with the number of players.

While the above construction directly implies an evolving secret sharing scheme for rigid evolving CNF/DNF formulas access structures, in Section 5.4 and Section 5.5, we give direct construc-

[6] An equivalent way to preserve monotonicity and rigidity is to assume that $\hat{C}_1(x_1, \ldots, x_{n_1}) = \hat{C}_2(x_1, \ldots, x_{n_1}, 0, \ldots, 0)$.

tions which are slightly better in terms of share size and/or assumptions. More in details,for the case of CNF formulas, we again rely on projective PRGs with *bounded* polynomial stretch m, and obtain instantiations from assumptions (i)-(iv) listed above with different trade-offs in terms of share size (see Table 1). For the case of DNF formulas, we only rely on OWFs and get a scheme with shares size $t_n \cdot (\lambda + 1)$, where t_n is the number of clauses in which the input associated to player n appears.

Domain extension. As our final contribution, we study the question of domain extension for evolving secret sharing schemes in the computational setting. Here, one starts with a secret sharing scheme supporting messages of size λ, for a rigid evolving access structure $\hat{\mathcal{A}}$, and the goal is to obtain a secret sharing scheme for the same access structure, but supporting messages of length $\ell \gg \lambda$.

In the non-evolving setting, this question was first studied by Krawczyk [26] for the case of threshold access structures. The main idea is to use a so-called information dispersal, which allows to divide the message into n fragments, in such a way that the message can be recovered from any $t \leq n$ fragments, while the size of each fragment is only ℓ/t (which is optimal). In other words, an information dispersal for the t-threshold access structure offers the same functionality of a secret sharing scheme for the same access structure, but without any privacy guarantee (which is the reason why the fragments can be shorter than the message). A simple example of information dispersal comes from Reed-Solomon codes: Parse the message μ into t blocks $\mu = (\mu_0, \ldots, \mu_{t-1})$, and interpret each block as an element of $\mathbb{GF}(q)$; if needed, the original message can be padded so that the message length ℓ is a multiple of the threshold t. Hence, let $f(X) = \mu_0 + \mu_1 \cdot X + \cdots + \mu_{t-1} \cdot X^{t-1}$ be the polynomial over $\mathbb{GF}(q)$, whose coefficients are the fragments of the message. The fragment γ_i assigned to party $i \in [n]$ is $f(i)$. Now, any subset of t parties can successfully reconstruct the polynomial, and thus recover the message. Moreover, the size of each fragment is $\log q = \ell/t$, which is optimal.

Given a secret sharing scheme (with domain $\{0,1\}^\lambda$) and an information dispersal for the t-threshold access structure, Krawczyk's compiler works as follows: First sample a random secret key $\kappa \in \{0,1\}^\lambda$ for a symmetric encryption scheme; then, encrypt the message $\mu \in \{0,1\}^\ell$. The share of party $i \in [n]$ is defined to be $\sigma'_i = (\sigma_i, \gamma_i)$, where σ_i is the i-th share of a secret sharing of the key κ, and γ_i is the i-th fragment of an information dispersal of the ciphertext γ. This results in shares of size $\ell/t + \lambda$, which is asymptotically optimal (as $\ell \to \infty$). In a follow-up work, Bèguin and Cresti [5] showed that the above construction still works assuming the underlying secret sharing scheme and information dispersal support an arbitrary access structure. Moreover, they observed that an information dispersal for access structure $\mathcal{A}$ can be obtained by dispersing the message using an information dispersal for the t-threshold access structure, where t is the minimum[7] size of an authorized set in $\mathcal{A}$.

In Section 6, we show that Krawczyk's compiler works unmodified even in the evolving setting. Naturally, this requires to assume an information dispersal for any rigid evolving access structure $\mathcal{A}$. To this end, we first show how to obtain an information dispersal for the evolving t-threshold access structure. Basically, this is an erasure code where one can disperse the message into a growing number of fragments (potentially infinite), with the guarantee that the message can be recovered from any fraction of t fragments. When $n = \mathsf{poly}(\lambda)$, an easy solution comes again using Reed-Solomon codes, i.e. we simply disperse the message using the above defined polynomial $f(X)$ over an exponentially large field $\mathbb{GF}(2^\lambda)$. This allows to accommodate an arbitrary polynomial number of users. A drawback of this solution is that it

[7]This solution is optimal in terms of minimizing the maximum share size. Bèguin and Cresti also propose simple variants that minimize the total size of the shares, but for simplicity we do not consider these variants in our paper.

requires an exponentially large field, and thus it is not very efficient in practice, as reconstruction takes quadratic (in t) time. This can be improved using more sophisticated techniques from coding theory. In particular, using so-called *digital fountains* [31] (e.g., Tornado codes [29], LT codes [28], or Raptor codes [38]), one can support a potentially infinite number of players with reconstruction time that is only linear (in t). To the best of our knowledge, this is the first application of rateless codes in cryptography, and thus we believe our work establishes an interesting connection with information theory.

Given an information dispersal for the evolving t-threshold access structure, one can obtain an information dispersal for any rigid evolving access structure $\hat{\mathcal{A}}$ by setting the threshold to the size of a minimum authorized subset in $\hat{\mathcal{A}}$, as proposed by Bèguin and Cresti [5]. A small caveat is that the latter requires the dealer to know the size of a minimal authorized subset in $\hat{\mathcal{A}}$; while this assumption is for free for some evolving access structures (e.g., dynamic threshold access structures, graphs access structures, and CNF formulas access structures), it is not always true in general (e.g., in the case of DNF formulas and monotone circuits access structures). We prove that this limitation is somewhat inherent for general access structures: whenever a rigid evolving access structure $\hat{\mathcal{A}}$ is such that the minimal size of an authorized subset goes from t_1 (when there are n_1 parties) to $t_2 < t_1$ (when there are $n_2 > n_1$ parties), no information dispersal for $\hat{\mathcal{A}}$ can be optimal in terms of share size (i.e., have maximum share size ℓ/t, where t is the minimum size of an authorized subset) without updating old shares.

1.3 Additional Related Work

Paskin-Cherniavsky [33] gives a secret sharing scheme for arbitrary evolving access structures with slightly better (but still exponential) share size (compared to [23]. Dutta *et al.* [16] consider a simple generalization of the evolving threshold access structure in which parties belong to different compartments, and each compartment has associated a threshold specifying the minimum size of an authorized subset in that compartment. Both of these constructions require that the dealer knows the access structure in advance.

Desmedt, Dutta, and Morozov [15] give an interpretation of evolving secret sharing from the lens of so-called evolving perfect hash families. This is useful in order to obtain schemes where the message space is a non-abelian group.

Komargodski and Paskin-Cherniavsky further show how to generically transform any secret sharing scheme for the evolving t-threshold access structure into a scheme which is *robust* [35], where the latter means that the message can be recovered even if some parties hand-in incorrect shares.

2 Preliminaries

2.1 Notation

Capital bold-face letters (such as $\mathbf{X}$) are used to denote probability distributions, small letters (such as x) to denote concrete values, calligraphic letters (such as $\mathcal{X}$) to denote sets, serif letters (such as A) to denote algorithms. For a string $x \in \{0,1\}^*$, we let $|x|$ be its length; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the cardinality of $\mathcal{X}$. When x is chosen uniformly from a set $\mathcal{X}$, we write $x \leftarrow\!\!\$\, \mathcal{X}$.

If $\mathcal{A}$ is a deterministic algorithm (modeled as a Turing machine), we write $y = \mathcal{A}(x)$ to denote a run of $\mathcal{A}$ on input x and output y; if $\mathcal{A}$ is randomized, we write $y \leftarrow\!\!\$\, \mathcal{A}(x)$ (or $y = \mathcal{A}(x;r)$) to denote a run of $\mathcal{A}$ on input x and (uniform) randomness r, and output y. An algorithm $\mathcal{A}$ is *probabilistic polynomial-time* (PPT) if $\mathcal{A}$ is randomized and for any input $x, r \in \{0,1\}^*$ the computation of $\mathcal{A}(x;r)$ terminates in a polynomial number of steps (in the input size).

We write $\mathsf{negl}(\lambda)$ to denote an arbitrary (unspecified) negligible function of the security parameter $\lambda \in \mathbb{N}$. Similarly, we write $\mathsf{poly}(\lambda)$ to denote an arbitrary (unspecified) polynomial function of the security parameter. We assume all algorithms take the security parameter (in unary) as input.

Let $\mathbf{X} = \{\mathbf{X}(\lambda)\}_{\lambda\in\mathbb{N}}$ and $\mathbf{Y} = \{\mathbf{Y}(\lambda)\}_{\lambda\in\mathbb{N}}$ be ensembles of random variables. We say that $\mathbf{X}$ and $\mathbf{Y}$ are computationally indistinguishable (denoted $\mathbf{X} \approx_c \mathbf{Y}$) if for all PPT adversaries A it holds that:

$$|\mathbb{P}[\mathsf{A}(\mathbf{X}(\lambda)) = 1] - \mathbb{P}[\mathsf{A}(\mathbf{Y}(\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

The notion of computational indistinguishability immediately extends to (ensembles of) interactive games $\mathbf{G}_{\mathsf{A}}(\lambda)$ featuring a PPT adversary and a challenger, where at the end of the game the adversary outputs a bit given its view.

2.2 Pseudorandom Generators

A PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^*$ is a polynomial-time algorithm G taking as input a uniformly random seed $\kappa \in \{0,1\}^\lambda$ and outputting a pseudorandom string $y \in \{0,1\}^*$. We recall the formal definition below.

Definition 1 (Security of PRG)**.** *A PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^*$ is secure if for every PPT adversary A we have:*

$$|\mathbb{P}[\mathsf{A}(\mathsf{G}(\kappa)) = 1] - \mathbb{P}[\mathsf{A}(y) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\kappa \leftarrow\!\!\$\ \{0,1\}^\lambda$ and $y \leftarrow\!\!\$\ \{0,1\}^*$.

2.3 Puncturable Pseudorandom Functions

A puncturable pseudorandom function (pPRF) with input space $\mathcal{X}$ and output space $\mathcal{Y}$ consists of three polynomial-time algorithms $\mathsf{pPRF} = (\mathsf{pPRF.Setup}, \mathsf{pPRF.Eval}, \mathsf{pPRF.Puncture})$ defined as follows:

- The setup algorithm $\mathsf{pPRF.Setup}(1^\lambda)$ takes as input the security parameter and outputs a key κ.
- The deterministic evaluation algorithm $\mathsf{pPRF.Eval}(\kappa, x)$ takes as input a key κ, and a value $x \in \mathcal{X}$, and outputs $y \in \mathcal{Y}$.
- The deterministic puncturing algorithm $\mathsf{pPRF.Puncture}(\kappa, x)$ takes as input a key κ, and a value $x \in \mathcal{X}$, and outputs a punctured key κ_x.

We now recall the definition of correctness and security of pPRFs.

Definition 2 (Correctness of pPRF)**.** *A puncturable pseudorandom function $\mathsf{pPRF} = (\mathsf{pPRF.Setup}, \mathsf{pPRF.pPRF.Eval}, \mathsf{pPRF.Puncture})$ with input space $\mathcal{X}$ and output space $\mathcal{Y}$ is correct if for all $\lambda \in \mathbb{N}$, for all $\kappa \in \mathsf{pPRF.Setup}(1^\lambda)$, and for all $x, x' \in \mathcal{X}$ such that $x \neq x'$, we have*

$$\mathbb{P}\big[\mathsf{pPRF.Eval}(\mathsf{pPRF.Puncture}(\kappa, x), x') = \mathsf{pPRF.Eval}(\kappa, x')\big] = 1.$$

Definition 3 (Security of pPRF)**.** *A puncturable pseudorandom function $\mathsf{pPRF} = (\mathsf{pPRF.Setup}, \mathsf{pPRF.Eval}, \mathsf{pPRF.Puncture})$ with input space $\mathcal{X}$ and output space $\mathcal{Y}$ is secure if for every PPT A, and for every $x \in \mathcal{X}$, we have*

$$|\mathbb{P}[\mathsf{A}(\kappa_x, \mathsf{pPRF.Eval}(\kappa, x)) = 1] - \mathbb{P}[\mathsf{A}(\kappa_x, y) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\kappa \leftarrow\!\!\$\ \mathsf{pPRF.Setup}(1^\lambda)$, $\kappa_x = \mathsf{pPRF.Puncture}(\kappa, x)$, and $y \leftarrow\!\!\$\ \mathcal{Y}$.

2.4 Secret-key Encryption

A secret-key encryption (SKE) scheme $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ over message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ consists of two polynomial-time algorithms specified as follows:

- The deterministic encryption algorithm $\mathsf{SKE.Enc}(\kappa, \mu)$ takes as input a secret key $\kappa \in \{0,1\}^\lambda$ and a message $\mu \in \mathcal{M}$, and outputs a ciphertext $\gamma \in \mathcal{C}$.
- The deterministic decryption algorithm $\mathsf{SKE.Dec}(\kappa, \gamma)$ takes as input a secret key $\kappa \in \{0,1\}^\lambda$ and a ciphertext $\gamma \in \mathcal{C}$, and outputs a message $\mu \in \mathcal{M}$.

The definition below says that a SKE scheme is correct if decryption inverts the encryption process (when using the same key).

Definition 4 (Correctness of SKE)**.** *We say that a secret-key encryption scheme* $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *over message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$ *is correct if for all* $\lambda \in \mathbb{N}$*, for all keys* $\kappa \in \{0,1\}^\lambda$*, and for all messages* $\mu \in \mathcal{M}$*, we have:*

$$\mathbb{P}[\mathsf{SKE.Dec}(\kappa, \mathsf{SKE.Enc}(\kappa, \mu)) = \mu] = 1.$$

The definition below captures security of SKE schemes in settings in which the adversary only obtains the encryption of a single message. Given a PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^\ell$, we can obtain such an SKE scheme over $\mathcal{M} = \mathcal{C} = \{0,1\}^\ell$ by letting $\mathsf{SKE.Enc}(\kappa, \mu) = \mathsf{G}(\kappa) \oplus \mu$ and $\mathsf{SKE.Dec}(\kappa, \gamma) = \mathsf{G}(\kappa) \oplus \gamma$.

Definition 5 (One-time security of SKE)**.** *We say that a secret-key encryption scheme* $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *over message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$ *is one-time secure if* $\{\mathbf{G}^{\mathsf{one\text{-}time}}_{\mathsf{SKE,A}}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{G}^{\mathsf{one\text{-}time}}_{\mathsf{SKE,A}}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$*, where the game* $\mathbf{G}^{\mathsf{one\text{-}time}}_{\mathsf{SKE,A}}(\lambda, b)$ *is specified as follows:*

- *The adversary picks two messages* $\mu_0, \mu_1 \in \mathcal{M}$ *and sends* (μ_0, μ_1) *to the challenger.*
- *The challenger picks* $\kappa \leftarrow\!\!\$\ \in \{0,1\}^\lambda$*, runs* $\gamma = \mathsf{SKE.Enc}(\kappa, \mu_b)$*, and sends* γ *to the adversary.*

2.5 Somewhere Statistically Binding Hash Functions

A somewhere statistically binding (SSB) hash function [20, 32] with block length ℓ_{block}, output length ℓ_{out}, and opening length ℓ_{open}, consists of four polynomial-time algorithms $\mathsf{SSB} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ specified as follows:

- The randomized setup algorithm $\mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{block}}}, m, i)$ takes as input the security parameter 1^λ, a block size $1^{\ell_{\mathsf{block}}}$, a message length $m \le 2^\lambda$, and an index $i \in [m]$, and outputs a key hk. Here, we assume that m and i are encoded in binary, i.e., the size of both m and $i \in [m]$ is bounded by $O(\log(m))$.
- The deterministic hashing algorithm $\mathsf{SSB.Hash}(\mathsf{hk}, (x_i)_{i\in[m']})$ takes as input a key hk and an input $(x_i)_{i\in[m']}$ (where $x_i \in \{0,1\}^{\ell_{\mathsf{block}}}$ and $m' \le m$), and outputs a hash $h \in \{0,1\}^{\ell_{\mathsf{out}}}$.
- The deterministic opening algorithm $\mathsf{SSB.Open}(\mathsf{hk}, (x_i)_{i\in[m']}, i)$ takes as input a key hk, an input $(x_i)_{i\in[m']}$ (where $x_i \in \{0,1\}^{\ell_{\mathsf{block}}}$ and $m' \le m$), and an index $i \in [m']$, and outputs an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$.
- The deterministic verification algorithm $\mathsf{SSB.Verify}(\mathsf{hk}, h, i, x_i, \pi_i)$ takes as input a key hk, a hash $h \in \{0,1\}^{\ell_{\mathsf{out}}}$, an index $i \in [m']$ (for $m' \le m$), an input $x_i \in \{0,1\}^{\ell_{\mathsf{block}}}$, and an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$, and outputs a bit.

Correctness of SSB hash functions says that honest openings always verify. As for security, SSB hash functions guarantee two properties, known as *index hiding* and *somewhere statistically binding*.

Definition 6 (Correctness of SSB hash functions)**.** *We say that a SSB hash function* $\mathsf{SSB} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ *is correct if* $\forall \lambda \in \mathbb{N}$, $\forall \ell_{\mathsf{block}} \in \mathbb{N}$, $\forall m \in \mathbb{N}$ *such that* $m \leq 2^\lambda$, $\forall m' \in \mathbb{N}$ *such that* $m' \leq m$, $\forall i^* \in [m]$, $\forall i \in [m']$, *and* $\forall (x_i)_{i\in[m']} \in \{0,1\}^{\ell_{\mathsf{block}} \cdot m'}$, *we have:*

$$\mathbb{P}\left[\mathsf{SSB.Verify}(\mathsf{hk}, h, i, x_i, \pi_i) = 1 : \begin{array}{c} \mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{block}}}, m, i^*), \\ h = \mathsf{SSB.Hash}(\mathsf{hk}, (x_i)_{i\in[m']}), \\ \pi_i = \mathsf{SSB.Open}(\mathsf{hk}, (x_i)_{i\in[m']}, i) \end{array}\right] = 1.$$

Definition 7 (Index hiding of SSB hash functions)**.** *We say that a SSB hash function* $\mathsf{SSB} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ *satisfies index hiding if for every PPT adversary* A, *for every* $\ell_{\mathsf{block}} \in \mathbb{N}$, *for every* $m \in \mathbb{N}$, *for every indexes* $i_0, i_1 \in [m]$, *we have:* $\left|\mathbb{P}\left[\mathsf{A}(1^\lambda, \mathsf{hk}_0) = 1\right] - \mathbb{P}\left[\mathsf{A}(1^\lambda, \mathsf{hk}_1) = 1\right]\right| \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{hk}_b = \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{block}}}, m, i_b)$ *for* $b \in \{0,1\}$.

Definition 8 (Somewhere statistically binding of SBB hash functions)**.** *We say that a SSB hash function* $\mathsf{SSB} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ *is somewhere statistically binding if for every* $\ell_{\mathsf{block}} \in \mathbb{N}$, *for every* $m \in \mathbb{N}$ *such that* $m \leq 2^\lambda$, *for every* $i \in [m]$, *we have:*

$$\mathbb{P}\left[\begin{array}{c} \nexists (h, x, x', \pi, \pi') \in \{0,1\}^{\ell_{\mathsf{out}} + 2\ell_{\mathsf{block}} + 2\ell_{\mathsf{open}}}\ s.t.\ x \neq x' \wedge \\ \mathsf{SSB.Verify}(\mathsf{hk}, h, i, x, \pi) = 1 \ \wedge \mathsf{SSB.Verify}(\mathsf{hk}, h, i, x', \pi') = 1 \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{block}}}, m, i)$.

In addition to the above properties, we focus on succinct and efficient SSB hash functions, which can be built from different assumptions such as DDH, ϕ-Hiding, DCR, and LWE [20, 32].

Definition 9 (Succinctness of SBB hash function)**.** *A SSB hash function* $\mathsf{SSB} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ *is succinct and efficient if the output length* ℓ_{out}, *the opening length* ℓ_{open}, *and the size of* hk *(output by* $\mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{block}}}, m, i)$*) are bounded by* $\mathsf{poly}(\lambda, \ell_{\mathsf{block}}, \log m)$.

In particular, we are interested in succinct SSB schemes where the maximum number of blocks m is exponential and where the block length is a single bit. When $m = 2^\lambda$ and $\ell_{\mathsf{block}} = 1$, we write $\mathsf{SSB.Setup}(1^\lambda, i)$ (instead of $\mathsf{SSB.Setup}(1^\lambda, 1^1, 2^\lambda, i)$). In this case, by definition of succinctness, the output length ℓ_{out}, the opening length ℓ_{open}, and the size of hk are bonded by $\mathsf{poly}(\lambda)$.

2.6 Indistinguishability Obfuscation

An indistinguishability obfuscator (iO) [4] is a PPT algorithm Obf that, upon input the security parameter 1^λ and a circuit Π, outputs an obfuscation $\mathsf{Obf}(1^\lambda, \Pi)$ of Π. An iO obfuscator must (i) preserve the functionality of the original circuit, and (ii) produce obfuscations of polynomial size w.r.t. to $|\Pi|$. Moreover, for every pair of functionally equivalent circuits Π_0, Π_1 (i.e., $\forall x \in \{0,1\}^*$, $\Pi_0(x) = \Pi_1(x)$ and $|\Pi_0| = |\Pi_1|$), $\mathsf{Obf}(1^\lambda, \Pi_0)$ and $\mathsf{Obf}(1^\lambda, \Pi_1)$ must be computationally indistinguishable.

Definition 10 (Indistinguishability obfuscation)**.** *Let* $\mathcal{C}$ *an ensemble of circuits. We say that a PPT algorithm* Obf *is an iO obfuscator if the following conditions are satisfied:*

Correctness. $\forall \lambda \in \mathbb{N}, \forall \Pi \in \mathcal{C}, \forall x \in \{0,1\}^*$, *we have that* $\Pi'(x) = \Pi(x)$ *where* $\Pi' \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, \Pi)$.

Polynomial slowdown. *For every* $\Pi \in \mathcal{C}$, *we have* $|\mathsf{Obf}(1^\lambda, \Pi)| \leq \mathsf{poly}(|\Pi|)$.

Indistinguishability. *For every pair of functionally-equivalent circuits* $\Pi_0, \Pi_1 \in \mathcal{C}$ *of equal size, for every PPT adversary* A, *we have*

$$\left|\mathbb{P}\Big[\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, \Pi_0)) = 1\Big] - \mathbb{P}\Big[\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, \Pi_1)) = 1\Big]\right| \leq \mathsf{negl}(\lambda).$$

2.7 RSA Assumption

We state the RSA assumption, as introduced by Rivest, Shamir and Adleman in [36].

Definition 11 (RSA assumption)**.** *We say that the (polynomial) RSA assumption holds if, for all PPT adversaries* A, *it holds that* $\mathbb{P}[\mathbf{G}^{\mathsf{rsa}}_{\mathsf{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$, *where the game* $\mathbf{G}^{\mathsf{rsa}}_{\mathsf{A}}(\lambda)$ *is defined as follows:*

- *The challenger samples two random* λ*-bit primes* p *and* q, *a random* λ*-bit prime* e, *and sets* $N = p \cdot q$. *Then, it picks* $y \leftarrow\!\!\$\ \mathbb{Z}^*_N$ *and lets* $z = y^e \bmod N$.
- *The adversary is given* (N, e, z) *and outputs* y'.
- *The game outputs* 1 *if and only if* $y' = y$.

The above is sometimes referred to as the RSA assumption with prime exponents, which differs from the standard RSA assumption in that the exponent e is a λ-bit random prime (instead of being a random value in $\mathbb{Z}^*_{\varphi(N)}$). However, because primes are $\Theta(1/\log N)$-dense in $\mathbb{Z}^*_{\varphi(N)}$, the two assumptions are equivalent.

RSA hard-core bits. Since the RSA assumption immediately implies a one-way function, we know that it admits a hard-core bit as proven by Goldreich and Levin [18]. Below, we state their result in the concrete case of the RSA assumption.

Theorem 1 (Goldreich-Levin Theorem [18])**.** *Under the RSA assumption, the following holds:*

$$(N, e, r, z, \langle r, y\rangle \bmod 2) \approx_c (N, e, r, z, \beta),$$

where $N = p \cdot q$ *for random* λ*-bit primes* p *and* q, e *is a random* λ*-bit prime,* $r \leftarrow\!\!\$\ \{0,1\}^\lambda$, $y \leftarrow\!\!\$\ \mathbb{Z}^*_N$, $z = y^e \bmod N$ *and* $\beta \leftarrow\!\!\$\ \{0,1\}$.

2.8 Projective Pseudorandom Generators

We review the recent notion of *projective* pseudorandom generators (pPRGs) from Applebaum *et al.* [1], which will be used as a tool in some of our constructions. For space reasons, we defer other auxiliary standard definitions to Section 2.

Intuitively, a projective PRG is a PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^m$ with the additional property that, given a master key msk and a subset $\mathcal{T} \subseteq [m]$, one can produce a *succinct*[8] projective key $\alpha_\mathcal{T}$ which can be used to recover the output bits $\mathsf{G}(\alpha)|_\mathcal{T}$, but reveals nothing about the other bits.

More formally, a projective PRG is a tuple of polynomial-time algorithms $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ specified as follows:

[8] Succinctness is a crucial requirement, as otherwise a projective key could just be $\mathsf{G}(\alpha)|_\mathcal{T}$.

- The randomized setup algorithm $\mathsf{pPRG.Setup}(1^\lambda, 1^m)$ takes as input the security parameter $\lambda \in \mathbb{N}$ and an output length parameter $m \in \mathbb{N}$, and outputs public parameters mpk along with a master secret key msk.
- The deterministic key generation algorithm $\mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T})$ takes as input the public parameters mpk, the master secret key msk, and a target set $\mathcal{T} \subseteq [m]$, and outputs a projective key $\alpha_{\mathcal{T}}$.
- The deterministic evaluation algorithm $\mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{\mathcal{T}}, \mathcal{T})$ takes as input the public parameters mpk, a projective key $\alpha_{\mathcal{T}}$, and a target set $\mathcal{T} \subseteq [m]$, and outputs a string $y \in \{0,1\}^{|\mathcal{T}|}$.

Abusing notation, we write $\mathsf{pPRG.Eval}(\mathsf{mpk}, \mathsf{msk})$ to denote the output of the PRG corresponding to $\mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{[m]}, [m])$ (i.e., when the target set $\mathcal{T}$ corresponds to the entire output length). A projective PRG is required to satisfy three properties. The first property is a correctness requirement saying that a projective key for target set $\mathcal{T}$ allows to learn the PRG output in the positions indexed by $\mathcal{T}$. The second property is a succinctness requirement saying that the size of a projective key for a target set $\mathcal{T}$ is significantly shorter than $|\mathcal{T}|$. The third property is a security requirement saying that an adversary obtaining a projective key for the union of different subsets $\mathcal{T}^*$ learn no information about the output of the PRG in the positions outside $\mathcal{T}^*$.

Definition 12 (Correctness of pPRGs)**.** *We say that* $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ *is correct if for all* $\lambda, m \in \mathbb{N}$*, all* $(\mathsf{mpk}, \mathsf{msk}) \in \mathsf{pPRG.Setup}(1^\lambda, 1^m)$*, all subsets* $\mathcal{T} \subseteq [m]$*, and all projective keys* $\alpha_{\mathcal{T}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T})$*, it holds that* $y_{\mathcal{T}} = \mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{\mathcal{T}}, \mathcal{T})$ *equals all of the bits of* $y = \mathsf{pPRG.Eval}(\mathsf{mpk}, \mathsf{msk})$ *indexed by the positions in* $\mathcal{T}$*.*

Definition 13 (Succinctness of pPRGs)**.** *We say that* $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ *is (fully) succinct if for all* $\lambda, m \in \mathbb{N}$*, all* $(\mathsf{mpk}, \mathsf{msk}) \in \mathsf{pPRG.Setup}(1^\lambda, 1^m)$*, and all subsets* $\mathcal{T} \subseteq [m]$*, it holds that the projective key* $\alpha_{\mathcal{T}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T})$ *has size* $|\alpha_{\mathcal{T}}| = \mathsf{poly}(\lambda, \log m)$*.*

Definition 14 (Robustness of pPRGs)**.** *We say that* $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ *is robust if for all PPT adversaries* A *it holds that* $\{\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{A}}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{A}}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$*, where the game* $\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{A}}(\lambda, b)$ *is defined as follows:*

- *Given* 1^λ*, the adversary sends* 1^m *and* $\mathcal{T}_1, \ldots, \mathcal{T}_q \subset [m]$ *to the challenger.*
- *The challenger runs* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{pPRG.Setup}(1^\lambda, 1^m)$ *and lets* $y_0 = y_{\overline{\mathcal{T}}}$ *and* $y_1 \leftarrow\!\!\$\ \{0,1\}^{|\overline{\mathcal{T}}|}$*, where* $\overline{\mathcal{T}} = [m] \backslash \mathcal{T}$*,* $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_q$*, and* $y_{\overline{\mathcal{T}}} = \mathsf{pPRG.Eval}(\mathsf{mpk}, \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \overline{\mathcal{T}}), \overline{\mathcal{T}})$*. Then, the challenger forwards* $(\mathsf{mpk}, (\alpha_{\mathcal{T}_i})_{i\in[q]}, y_b)$ *to* A*, where* $\alpha_{\mathcal{T}_i} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T}_i)$*.*

2.8.1 Unbounded Polynomial Stretch

For some of our applications, we will require a stronger variant of projective PRGs in which the output length is an unbounded polynomial $m = \mathsf{poly}(\lambda)$. For concreteness, we define this variant below.

Definition 15 (pPRGs with unbounded stretch)**.** *We say that* $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ *has* unbounded polynomial stretch *if algorithm* Setup *takes only the security parameter* $\lambda \in \mathbb{N}$ *as input, whereas algorithms* KeyGen *and* Eval *take as input target sets* $\mathcal{T}$ *of arbitrary polynomial size* $|\mathcal{T}| = \mathsf{poly}(\lambda)$*.*

The definitions of correctness, succinctness and robustness can be easily adapted to the case of projective PRGs with unbounded polynomial stretch. In particular, correctness is immediate, while succinctness still requires that the size of the projective keys $\alpha_{\mathcal{T}}$ are $\mathsf{poly}(\lambda)$, whereas the running time of algorithms KeyGen and Eval are $\mathsf{poly}(\lambda, |\mathcal{T}|)$. We remark that, when the stretch is unbounded, the master public key mpk is always succinct (i.e., of size $\mathsf{poly}(\lambda)$), as the setup algorithm does not depend on m.

As for robustness, the security game remains unchanged as the adversary can already specify the output length 1^m of the challenge. Note the adversary has to commit to 1^m and to the subsets $\mathcal{T}_1, \cdots, \mathcal{T}_q$ before receiving the master public key mpk. This flavor of "selective" security is sufficient for our applications.

Outputting blocks. Sometimes, it is convenient to think of the pPRG output as a sequence of t blocks of size λ. In such a case, the key generation takes as input subsets $\mathcal{T} \subseteq [t]$ and generates projective keys corresponding to the blocks indexed by the positions in $\mathcal{T}$; the evaluation algorithm is modified analogously. The latter can be obtained by mapping $\mathcal{T}$ into $\mathcal{T}' \subseteq [t \cdot \lambda]$, where $\mathcal{T}'$ consists of the set of all location that fall inside the blocks whose indexes are in $\mathcal{T}$.

2.8.2 Instantiations

The following theorem summarizes known constructions of projective PRGs under a variety of assumptions.

Theorem 2 ([1])**.** *There exist constructions of projective PRGs from the following assumptions and with the following parameters:*

- *Under the RSA assumption, with* unbounded *polynomial stretch, and with master public keys and projective keys of size* $\mathsf{poly}(\lambda)$*.*
- *Assuming indistinguishability obfuscation and somewhere statistically binding hash functions, with* unbounded *polynomial stretch, and with projective keys of size* $\mathsf{poly}(\lambda)$ *(and empty master public keys).*
- *Under the DDH and the BDDH assumption, with* bounded *polynomial stretch, and with master public keys of size* $m^2 \cdot \mathsf{poly}(\lambda)$ *and projective keys of size* $O(\lambda)$*. In the second construction the master public key is reusable (i.e., it is independent of the master secret key).*
- *Under the LWE assumption, with* bounded *polynomial stretch, and with master public keys of size* $m \cdot \mathsf{poly}(\lambda)$ *and projective keys of size* $O(\lambda)$*.*

We remark that Applebaum *et al.* [1] actually prove a slightly different statement about the constructions based on RSA and on obfuscation. In particular, they prove that the first construction achieves sub-exponential stretch under the sub-exponential RSA assumption, and that the second construction achieves *bounded* polynomial stretch. Nevertheless, it is easy to adapt these constructions and show that they indeed achieve *unbounded* polynomial stretch under polynomial hardness.

2.8.3 Projective PRGs with Unbounded Polynomial Stretch

Instantiation from RSA. We show that the construction of pPRGs from [1], based on the RSA assumption, can easily be adapted to yield unbounded polynomial stretch under the polynomial RSA assumption.

Construction 1

Let $\bar{m} = 2^{\lambda/3}$ and $k = \lambda \cdot (\lambda + \log \bar{m})$. Consider the following projective PRG $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ with unbounded polynomial stretch:

- The setup algorithm $\mathsf{pPRG.Setup}(1^\lambda)$ outputs $\mathsf{msk} = (p, q)$ and $\mathsf{mpk} = (N, u, r, \rho)$, where p and q are random λ-bit primes, $N = p \cdot q$, $u \leftarrow\!\!\$\ \mathbb{Z}_N^*$, $r \leftarrow\!\!\$\ \{0,1\}^\lambda$, and $\rho = (\rho_0, \ldots, \rho_{k-1})$ for $\rho_i \leftarrow\!\!\$\ \mathbb{GF}(2^\lambda)$. We think of ρ as a degree-k polynomial $\Psi_\rho(X) = \sum_i \rho_i \cdot X^i$ over $\mathbb{GF}(2^\lambda)$, and let $x_1, x_2, \ldots$ be a canonical set of distinct evaluation points in $\mathbb{GF}(2^\lambda)$.
- The key generation algorithm $\mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T})$ proceeds as follows:
  - Let $|\mathcal{T}| = t$. Evaluate the polynomial $\Psi_\rho(X)$ using the evaluation points $x_1, \ldots, x_{t \cdot k}$, obtaining $t \cdot k$ integers of size λ, and, for each $i \in [t]$, define e_i to be the first prime in the i-th block of k integers.
  - Output the projective key $\alpha_{\mathcal{T}} = u^{\prod_{j \in \mathcal{T}} 1/e_j} \bmod N$, where the computations in the exponent are modulo $\varphi(N) = (p-1) \cdot (q-1)$.
- The evaluation algorithm $\mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{\mathcal{T}}, \mathcal{T})$ proceeds as follows for each $i \in \mathcal{T}$:
  - Let $|\mathcal{T}| = t$. Evaluate the polynomial $\Psi_\rho(X)$ using the evaluation points $x_1, \ldots, x_{t \cdot k}$, obtaining $t \cdot k$ integers of size λ, and, for each $i \in [t]$, define e_i to be the first prime in the i-th block of k integers.
  - Compute $y'_i = \alpha_{\mathcal{T}}^{\prod_{j \in \mathcal{T} \setminus \{i\}} e_j} \bmod N$ and output $y_i = \langle r, y'_i \rangle \bmod 2$.

The full output of the projective PRG is defined to be $(y_1, \ldots, y_{\bar{m}})$ where $y_i = \langle r, y'_i \rangle$ for $y'_i = u^{1/e_i}$ and for all $i \in [\bar{m}]$; note that $\bar{m}$ is exponential, and indeed the key generation and evaluation algorithms never need to generate all of the primes $(e_1, \ldots, e_{\bar{m}})$. Correctness follows because:

$$y'_i \equiv \alpha_{\mathcal{T}}^{\prod_{j \in \mathcal{T} \setminus \{i\}} e_j} \equiv \left(u^{\prod_{j \in \mathcal{T}} 1/e_j}\right)^{\prod_{j \in \mathcal{T} \setminus \{i\}} e_j} \equiv u^{1/e_i} \pmod N.$$

Note that, by the prime number theorem, the probability that each of the values e_i computed during key generation and evaluation is not a prime, or that the primes are not distinct, is $\exp(-\Omega(\lambda))$; for simplicity, we just ignore these events when analyzing the construction.[9] Succinctness follows because, by inspection, both the master public key and the projective keys have size $\mathsf{poly}(\lambda)$. As for security, we have the following theorem:

Theorem 3. *Under the RSA assumption (Definition 11), the projective PRG* pPRG *described in Construction 1 is robust.*

Proof. By contradiction, let B be any PPT adversary that breaks the robustness of pPRG with probability non-negligible probability. Denote by $(1^m, \mathcal{T}_1, \ldots, \mathcal{T}_q)$ the values output by B at the beginning of the robustness game, where $m = \mathsf{poly}(\lambda)$ and $|\mathcal{T}_i| = \mathsf{poly}(\lambda)$ for all $i \in [q]$. Also, let $\overline{\mathcal{T}} = [m] \setminus \bigcup_{i \in [q]} \mathcal{T}_i = \{i_1, \ldots, i_{\bar{t}}\}$ for some $\bar{t}$ such that $0 \leq \bar{t} < m$.

For an index $j \in [0, \bar{t}]$, consider the hybrid experiment $\mathbf{H}_j(\lambda)$ in which we modify the challenge given to the attacker B as follows:

[9] Strictly speaking, this means that the construction does not achieve perfect correctness, but instead correctness only holds with overwhelming probability over the randomness of the setup algorithm.

- For each of the sets $\mathcal{T}_1, \ldots, \mathcal{T}_q$, the projective keys $(\alpha_{\mathcal{T}_i})_{i\in[q]}$ are computed as defined in Construction 1.
- For each $h \leq j$, the challenge bits y_{i_h} are computed as defined in Construction 1.
- For each $h > j$, the challenge bits y_{i_h} are sampled uniformly at random.

Clearly,

$$\{\mathbf{H}_0(\lambda)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{B}}(\lambda,0)\}_{\lambda\in\mathbb{N}} \qquad \{\mathbf{H}_{\bar{t}}(\lambda)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{B}}(\lambda,1)\}_{\lambda\in\mathbb{N}}.$$

Thus, by the hybrid argument, there exists an index $j \in [\bar{t}]$ such that:

$$|\mathbb{P}[\mathbf{H}_j(\lambda) = 1] - \mathbb{P}[\mathbf{H}_{j-1}(\lambda) = 1]| \geq \epsilon.$$

where ϵ is non-negligible. We now construct an adversary A (using B) that breaks security of the hard-core bit for the RSA one-way function:

- The input of A is a tuple (N, e, z, r, β), where $N = p \cdot q$ for two random λ-bit primes, e is a random λ-bit prime, $z = y^e \bmod N$ for random $y \leftarrow\!\!\$\ \mathbb{Z}_N^*$, $r \leftarrow\!\!\$\ \{0,1\}^\lambda$, and β is either the hard-core bit of (r, y) (i.e., $\beta = \langle r, y\rangle \bmod 2$) or a uniformly random bit.
- Run $\mathsf{B}(1^\lambda)$, obtaining the tuple $(1^m, \mathcal{T}_1, \ldots, \mathcal{T}_q)$.
- Simulate the master public key for the projective PRG as follows:
  - Let $k = \lambda \cdot (\lambda + \log m)$, then sample k random integers and replace the first integer with e from the challenge. Hence, using polynomial interpolation, find a random seed $\rho = (\rho_0, \ldots, \rho_{k-1})$ that generates the above integers as the i_j-th sequence of integers.
  - Compute $u = z^{\prod_{i\in[m]:i\neq i_j} e_i}$, where e_i are the primes associated to the polynomial corresponding to the seed ρ determined in the previous step.
  - Set $\mathsf{mpk} = (N, u, r, \rho)$, and return mpk to the adversary B.
- Simulate the projective keys $\alpha_{\mathcal{T}_i}$, for each $i \in [q]$, by letting:

$$\alpha_{\mathcal{T}_i} = z^{\prod_{h\notin\mathcal{T}_i:h\neq i_j} e_h}.$$

- Simulate the output bits y_{i_h}, for each $h \in [\bar{t}]$, by letting

$$y_{i_h} = \begin{cases} \langle r, y'_{i_h}\rangle \bmod 2 & \text{If } h < j \\ \beta & \text{If } h = j \\ \text{uniform} & \text{If } h > j, \end{cases}$$

  where $y'_{i_h} = z^{\prod_{i\in[m]:i\neq i_j,i_h} e_i}$.
- Send $(\mathsf{mpk}, (\alpha_{\mathcal{T}_i})_{i\in[q]}, (y_{i_h})_{j\in[\bar{t}]})$ to B and output whatever B outputs.

For the analysis, we note that:

- The simulation of the public key is perfect. Indeed, N and e are chosen randomly and the seed ρ is random subject to $e_j = e$. Thus, N and ρ are random. Moreover, since y is uniform, $u = y^{\prod_{i\in[m]} e_i}$ is a permutation, and thus the value u simulated by the reduction is uniform over $\mathbb{Z}_N^*$. Note that $y = u^{\prod_{i\in[m]} 1/e_i}$, although the reduction does not know $1/e_i$.

- For each $i \in [q]$, the simulation of the projective keys $\alpha_{\mathcal{T}_i}$ is perfect, as $z = y^e = y^{e_{i_j}}$ and thus $\alpha_{\mathcal{T}_i} = y^{\prod_{h \notin \mathcal{T}_i} e_h} = (u^{\prod_{i \in [m]} 1/e_i})^{\prod_{h \notin \mathcal{T}_i} e_h} = u^{\prod_{h \in \mathcal{T}_i} 1/e_h}$.
- For each $h \in [t]$, the simulation of the output bits y_{i_h} is perfect, as $z = y^e = y^{e_{i_j}}$ and thus:
  - If $h < j$, we have that $y'_{i_h} = y^{\prod_{i \in [m]: i \neq i_h} e_i} = (u^{\prod_{i \in [m]} 1/e_i})^{\prod_{i \in [m]: i \neq i_h} e_i} = u^{\prod_{i \in [m]: i \neq i_h} e_i}$, and $y_{i_h} = \langle r, y'_{i_h} \rangle \bmod 2$ as defined in both hybrid experiments.
  - If $h = j$, we have that $y_{i_h} = \beta$ is either equal to $\langle r, y \rangle = \langle r, y'_{i_j} \rangle$ (as defined in $\mathbf{H}_j(\lambda)$) or uniformly random (as defined in $\mathbf{H}_{j+1}(\lambda)$).
  - If $h > j$, we have that y_{i_h} is uniform as defined in both hybrids experiments.

We conclude that A breaks security of the hard-core bit for RSA with non-negligible probability. By Theorem 1 this contradicts the RSA assumption, and thus concludes the proof. □

Instantiation from Obfuscation. We show that the construction of projective PRGs from [1] based on obfuscation, can easily be adapted to yield unbounded polynomial stretch.

Construction 2

Let Obf be an iO obfuscator, $\mathsf{pPRF} = (\mathsf{pPRF.Setup}, \mathsf{pPRF.Eval}, \mathsf{pPRF.Puncture})$ be a puncturable PRF with input space $\{0,1\}^\lambda$ and output space $\{0,1\}$, $\mathsf{SSB} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ be a SSB hash function. Consider the following projective PRG $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ with unbounded polynomial stretch:

- The setup algorithm $\mathsf{pPRG.Setup}(1^\lambda)$ outputs $\mathsf{msk} = \kappa$ and $\mathsf{mpk} = \mathsf{hk}$ where $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1)$ and $\kappa \leftarrow\!\!\$\ \mathsf{pPRF.Setup}(1^\lambda)$.
- The key generation algorithm $\mathsf{pPRF.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T})$ proceeds as follows:
  - Compute $h_\mathcal{T} = \mathsf{SSB.Hash}(\mathsf{hk}, (v_i)_{i \in |\mathcal{T}|})$ where $v_i = 1$ if $i \in \mathcal{T}$; otherwise $v_i = 0$.
  - Compute $\tilde{\Pi}_\mathcal{T} \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, \Pi_{\mathsf{hk}, h_\mathcal{T}, \kappa})$ where the circuit $\Pi_{\mathsf{hk}, h_\mathcal{T}, \kappa}$ is defined as follows:

    $\Pi_{\mathsf{hk}, h_\mathcal{T}, \kappa}(i, \pi_i)$

    If $\mathsf{SSB}, \mathsf{Verify}(\mathsf{hk}, h_\mathcal{T}, i, 1, \pi_i) = 1$: **return** $\mathsf{pPRF.Eval}(\kappa, i)$

    return $\perp$

    The circuit $\Pi_{\mathsf{hk}, h_\mathcal{T}, \kappa}$ is padded to match the size $\max\{\Pi_{\mathsf{hk}, h_\mathcal{T}, \kappa}, \Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_\mathcal{T}, \kappa_{i_{j+1}}}\} = \mathsf{poly}(\lambda)$, where $\Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_\mathcal{T}, \kappa_{i_{j+1}}}$ is defined in the security proof below. (Recall that $i_{j+1} \leq 2^\lambda$; hence, it can be represented using λ bits.)

  Finally, it outputs $\alpha_\mathcal{T} = \tilde{\Pi}_\mathcal{T}$.
- The evaluation algorithm $\mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_\mathcal{T}, \mathcal{T})$ proceeds as follows for each $i \in \mathcal{T}$:
  - Compute $\pi_i = \mathsf{SSB.Open}(\mathsf{hk}, (v_j)_{j \in \mathcal{T}}, i)$ where $v_j = 1$ if $j \in \mathcal{T}$; otherwise $v_j = 0$.
  - Execute $y_i = \tilde{\Pi}_\mathcal{T}(i, \pi_i)$.

  Finally, it outputs $y = (y_1, \ldots, y_{|\mathcal{T}|})$.

Correctness is immediate. Succinctness follows by inspection, as the size of both the master public key and of the projective keys is $\mathsf{poly}(\lambda)$. As for security, we have the following theorem:

Theorem 4. *Assuming* Obf *is secure (Definition 10),* SSB *is somewhere statistically binding (Definition 8) and satisfies index hiding (Definition 7), and* pPRF *is secure (Definition 3), then the projective PRG described in Construction 2 is robust.*

Proof. Without loss generality, let 1^m be the output length chosen by the adversary in the robustness game and $\overline{\mathcal{T}} = \{i_1, \ldots, i_{\bar{t}}\} = [m] \setminus \mathcal{T}$. Consider the following hybrid experiments:

$\mathbf{H}_{0,0}(\lambda)$**:** This experiment is identical to $\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{A}}(\lambda, 0)$.

$\mathbf{H}_{j,0}(\lambda)$ **for** $j \in [\bar{t}]$**:** Identical to $\mathbf{H}_{j-1,0}(\lambda)$, except that the challenger sets $y_0 = y_{\overline{\mathcal{T}}} = y' || y''$ where $y' \leftarrow\!\!\$\ \{0,1\}^j$ and y'' are the last $\bar{t} - j$ bits of $\mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{\overline{\mathcal{T}}}, \overline{\mathcal{T}})$ for $\alpha_{\overline{\mathcal{T}}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \overline{\mathcal{T}})$.

$\mathbf{H}_{j,1}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,0}(\lambda)$ except that the challenger computes $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, i_{j+1})$.

$\mathbf{H}_{j,2}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,1}(\lambda)$, except that the challenger, for every $i \in [q]$, computes $\alpha_{\mathcal{T}_i} = \tilde{\Pi}_{\mathcal{T}_i} \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, \Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_{\mathcal{T}_i}, \kappa_{j+1}})$ where $i_{j+1} \in \overline{\mathcal{T}}$, $\kappa_{i_{j+1}} = \mathsf{pPRF.Puncture}(\kappa, i_{j+1})$, $\gamma = \mathsf{pPRF.Eval}(\kappa, i_{j+1})$, and the circuit $\Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_{\mathcal{T}}, \kappa_{i_{j+1}}}$ is defined as follows:[10]

$\Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_{\mathcal{T}}, \kappa_{i_{j+1}}}(i, \pi_i)$

If $i = i_{j+1}$: **return** γ

If $\mathsf{SSB.Verify}(\mathsf{hk}, h_{\mathcal{T}}, i, 1, \pi_i) = 1$: **return** $\mathsf{pPRF.Eval}(\kappa_{i_{j+1}}, i)$

return $\perp$

$\mathbf{H}_{j,3}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,2}(\lambda)$, except that the challenger sets $\gamma = \perp$ in $\Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_{\mathcal{T}_i}, \kappa_{j+1}}$ for every $i \in [q]$.

$\mathbf{H}_{j,4}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,3}(\lambda)$, except that the challenger sets $y_0 = y_{\overline{\mathcal{T}}} = y' || y''$ where $y' \leftarrow\!\!\$\ \{0,1\}^{j+1}$ and y'' are the last $\bar{t} - j - 1$ bits of $\mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{\overline{\mathcal{T}}}, \overline{\mathcal{T}})$ for $\alpha_{\overline{\mathcal{T}}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \overline{\mathcal{T}})$.

$\mathbf{H}_{j,5}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,4}(\lambda)$, except that the challenger sets $\gamma = \mathsf{pPRF.Eval}(\kappa, i_{j+1})$ in $\Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_{\mathcal{T}_i}, \kappa_{j+1}}$ for every $i \in [q]$.

$\mathbf{H}_{j,6}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,5}(\lambda)$, except that the challenger, for every $i \in [q]$, computes $\alpha_{\mathcal{T}_i} = \tilde{\Pi}_{\mathcal{T}_i} \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, \Pi_{\mathsf{hk}, h_{\mathcal{T}_i}, \kappa})$ where circuit $\Pi_{\mathsf{hk}, h_{\mathcal{T}}, \kappa}$ is defined as in Construction 2.

$\mathbf{H}_{j,7}(\lambda)$ **for** $j \in [\bar{t} - 1]$**:** Identical to $\mathbf{H}_{j,6}(\lambda)$, except that the challenger computes $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1)$. This experiment is identical to $\mathbf{G}^{\mathsf{rob}}_{\mathsf{pPRG},\mathsf{A}}(\lambda, 1)$.

Lemma 1. *For every* $j \in [0, \bar{t} - 1]$*, it holds that* $\{\mathbf{H}_{j,0}(\lambda)\}_{\lambda \in \mathbb{N}} \approx_c \{\mathbf{H}_{j,1}(\lambda)\}_{\lambda \in \mathbb{N}}$.

Proof. The lemma follows directly by the index hiding property of SSB (Definition 7). □

[10] Recall that the $\Pi_{\mathsf{hk}, h_{\mathcal{T}}, \kappa_{i_{j+1}}}$ of Construction 2 is padded to match the size $\max\{\Pi_{\mathsf{hk}, h_{\mathcal{T}}, \kappa_{i_{j+1}}}, \Pi^{\gamma, i_{j+1}}_{\mathsf{hk}, h_{\mathcal{T}}, \kappa_{i_{j+1}}}\}$. This is required during the proof in order to use the security of the iO obfuscator Obf.

Lemma 2. *For every $j \in [0, \bar{t}-1]$, it holds that $\{\mathbf{H}_{j,1}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{j,2}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

Proof. It is easy to see that $\Pi_{\mathsf{hk},h_{\mathcal{T}_i},\kappa}$ (of $\mathbf{H}_{j,1}(\lambda)$) and $\Pi^{\gamma,i_{j+1}}_{\mathsf{hk},h_{\mathcal{T}_i},\kappa_{i_{j+1}}}$ (of $\mathbf{H}_{j,2}(\lambda)$) are functionally-equivalent for every $i \in [q]$. Hence, the lemma follows directly by the security of the iO obfuscator Obf (Definition 10). □

Lemma 3. *For every $j \in [0, \bar{t}-1]$, it holds that $\{\mathbf{H}_{j,2}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{j,3}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

Proof. The only input on which $\Pi^{\gamma,i_{j+1}}_{\mathsf{hk},h_{\mathcal{T}_i},\kappa_{i_{j+1}}}$ (of $\mathbf{H}_{j,2}(\lambda)$) and $\Pi^{\perp,i_{j+1}}_{\mathsf{hk},h_{\mathcal{T}_i},\kappa_{i_{j+1}}}$ (of $\mathbf{H}_{j,3}(\lambda)$) may differ is one for which $(i^*,\pi^*) = (i_{j+1},\pi^*)$. However, since SSB is somewhere statistically binding (Definition 8), there is no valid opening π^* for values $v_{i_{j+1}} = 1$. Hence, both circuits output $\perp$ on input $(i^*,\pi^*) = (i_{j+1},\pi^*)$. Since the circuits are functionally-equivalent, the lemma follows directly by the security of the iO obfuscator Obf (Definition 10). □

Lemma 4. *For every $j \in [0, \bar{t}-1]$, it holds that $\{\mathbf{H}_{j,3}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{j,4}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

Proof. The lemma follows directly by the security of pPRF (Definition 3). □

Lemma 5. *For every $j \in [0, \bar{t}-1]$, it holds that $\{\mathbf{H}_{j,4}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{j,5}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

Proof. The lemma follows by an identical argument to that in the proof of Lemma 3. □

Lemma 6. *For every $j \in [0, \bar{t}-1]$, it holds that $\{\mathbf{H}_{j,5}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{j,6}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

Proof. The lemma follows by an identical argument to that in the proof of Lemma 2. □

Lemma 7. *For every $j \in [0, \bar{t}-1]$, it holds that $\{\mathbf{H}_{j,6}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{j,7}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

Proof. The lemma follows by an identical argument to that in the proof of Lemma 1. □

The theorem now follows by combining the above lemmas. □

3 Computational Evolving Secret Sharing

Let $n \in \mathbb{N}$, and denote by $[n] = \{1,\ldots,n\}$. A collection of subsets $\mathcal{A} \subseteq 2^{[n]}$ is *monotone* if for every $\mathcal{I} \in \mathcal{A}$, with $\mathcal{I} \subseteq \mathcal{I}'$, it holds that $\mathcal{I}' \in \mathcal{A}$.

Definition 16 (Access structure)**.** *An access structure $\mathcal{A} \subseteq 2^{[n]}$ is a monotone collection of non-empty subsets. Subsets in $\mathcal{A}$ are called* qualified*, whereas subsets not in $\mathcal{A}$ are called* unqualified*.*

If $\mathcal{A}$ is an access structure, then a subset $\mathcal{B} \in \mathcal{A}$ is *minimal* if $\mathcal{B}' \notin \mathcal{A}$ whenever $\mathcal{B}' \subset \mathcal{B}$. We will typically assume $\mathcal{A}$ only consists of minimal authorized subsets, which we sometimes refer to as the minimal representation form.

Standard secret sharing schemes are typically defined in a setting where the number of parties n is fixed. In evolving secret sharing [23], instead, parties arrive one by one and thus the value n represents the number of parties currently in the system. In the information-theoretic setting, the number n can grow up to infinity; in the computational setting, instead, $n = \mathsf{poly}(\lambda)$. We denote by $\mathcal{A}_n$ the access structure when there are n parties in the system. We stress, that the dealer does not know the value n, neither it knows the access structure $\mathcal{A}_n$ before the n-th party enters the system.

Definition 17 (Evolving access structure)**.** *Let $n \in \mathbb{N}$, with $n(\lambda) = \mathsf{poly}(\lambda)$. An evolving access structure $\mathcal{A} = \{\mathcal{A}_n\}$ is a monotone collection of subsets $\mathcal{A}_n \subseteq 2^{[n]}$ such that, for every $n \in \mathbb{N}$, it holds that $\mathcal{A}_n \subseteq \mathcal{A}_{n+1}$.*

We are now ready to define evolving secret sharing schemes. A secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M}$ for an evolving access structure $\mathcal{A}$ consists of two polynomial-time algorithms specified as follows:

- The randomized sharing algorithm $\mathsf{SS.Share}(\mu, (\sigma_i)_{i\in[n-1]}, \mathcal{A}_n)$ takes as input a message $\mu \in \mathcal{M}$, a collection of $n-1$ shares $\sigma_1, \ldots, \sigma_{n-1} \in \mathcal{S}$, and an access structure $\mathcal{A}_n$, and outputs the share σ_n for the n-th party.
- The deterministic reconstruction algorithm $\mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{I}}, \mathcal{I}, \mathcal{A}_n)$ takes as input a collection of shares $(\sigma_i)_{i\in\mathcal{I}}$, along with a subset $\mathcal{I} \subseteq 2^{[n]}$ of the parties, and outputs a message $\mu \in \mathcal{M}$.

The share size $s(n, \ell, \lambda)$ of party n in a secret sharing scheme for evolving access structure $\mathcal{A}$ is defined as the maximum of $|\sigma_n|$ over all messages of length $\ell = \ell(n)$ and over all possible previous assignments $\sigma_1, \ldots, \sigma_{n-1}$. Whenever $s(n, \ell, \lambda) = \mathsf{poly}(\lambda)$, we say that SS is *succinct*.

Remark 1 (On representing the access structure)**.** *When dealing with efficiency of computational secret sharing schemes, it is important to define how an evolving access structure is represented. In particular, we can associate to each access structure $\mathcal{A}_n$ over n parties a boolean function $f_n : \{0,1\}^n \to \{0,1\}$ such that $f(x) = 1$ if and only if the set $\mathcal{I}_x$ consisting of all the parties in $[n]$ for which $x_i = 1$ is such that $\mathcal{I}_x \in \mathcal{A}_n$. Hence, we assume there is a universal (polynomial-time computable) representation model $\mathsf{U} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ such that $\mathsf{U}(\Pi_n, x) = f_n(x)$ for all $n \geq 1$ and for all $x \in \{0,1\}^n$, where Π_n is a program representing the function f_n. For simplicity, in the rest of the paper, we implicitly assume that the sharing and reconstruction algorithm take as input the program Π_n representing the function f_n corresponding to the access structure $\mathcal{A}_n$; sometimes, we abuse notation and write $\mathcal{A}_n$ instead of the program Π_n.*

3.1 Defining Computational Privacy

Evolving secret sharing schemes are required to satisfy the following two properties. The first property is a correctness requirement saying that, at any point in time, qualified subsets of parties can recover the message. The second property is a security requirement saying that, at any point in time, a computationally bounded adversary knowing the shares corresponding to an unqualified subset of parties obtains no information about the message.

Definition 18 (Correctness of evolving secret sharing)**.** *We say that a secret sharing scheme SS over message space $\mathcal{M}$ for the evolving access structure $\mathcal{A}$ is correct if for every message $\mu \in \mathcal{M}$, for every number of parties $n \geq 1$, and for every qualified subset $\mathcal{I} \in \mathcal{A}_n$ it holds that*

$$\mathbb{P}\left[\mu = \mu' : \begin{array}{c} \forall i \leq n : \sigma_i \leftarrow\!\!\$\ \mathsf{SS.Share}(\mu, (\sigma_j)_{j<i}, \mathcal{A}_i) \\ \mu' = \mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{I}}, \mathcal{I}) \end{array}\right] = 1.$$

Definition 19 (Privacy of evolving secret sharing)**.** *We say that a secret sharing scheme SS over message space $\mathcal{M}$ for the evolving access structure $\mathcal{A}$ is computationally private if $\{\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$, where the game $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$ is defined as follows:*

- *The adversary chooses a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$, an integer $n \geq 1$, and an unqualified subset $\mathcal{U} \notin \mathcal{A}_n$, and forwards them to the challenger.*
- *The challenger computes $\sigma_i \leftarrow\!\!\$\ \mathsf{SS.Share}(\mu_b, (\sigma_j)_{j<i}, \mathcal{A}_i)$ for all $i \leq n$, and forwards $(\sigma_i)_{i\in\mathcal{U}}$ to A.*

3.2 Rigid Access Structures

We remark that our definition of evolving access structures is less stringent than previous definitions considered in the literature (see, e.g., [23]). In particular, previous definitions require that if at some point there is an unqualified subset $\mathcal{U} \subseteq 2^{[n]}$, where n is the current number of parties, then the set $\mathcal{U}$ will always remain unqualified when more parties are added to the evolving access structure. In contrast, Definition 17 potentially allows sets that previously were unqualified to become qualified at a later point in time (without necessarily involving new players). The theorem below states that Definition 17 is impossible to achieve unless the dealer is allowed to update the shares.

Theorem 5. *Let $\mathcal{A}$ be an evolving access structure such that there exist indexes n_1, n_2, with $n_1 < n_2$, along with a subset $\mathcal{U} \in 2^{[n_1]}$ which satisfy the following conditions: (i) $\mathcal{U} \notin \mathcal{A}_{n_1}$; (ii) $\mathcal{U} \in \mathcal{A}_{n_2}$. Then, any computationally private secret sharing scheme for $\mathcal{A}$ requires to update the shares of party n_1 when party n_2 enters the system.*

Proof. Let $\mathcal{A}$, n_1, and n_2 be as in the statement of the theorem. Fix an arbitrary message $\mu \in \mathcal{M}$, and denote by $(\sigma_i)_{i\in[n_2]}$ the shares obtained by sharing μ among n_2 parties. Condition (ii) on the access structure $\mathcal{A}$, along with the correctness property of SS, imply that $\mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{U}},\mathcal{U})$ outputs μ with probability one over the randomness of the sharing algorithm $\mathsf{SS.Share}$.

Now, consider the adversary A that plays the computational privacy game by choosing messages $\mu_0 = \mu$ and $\mu_1 = \mu' \neq \mu$, number of parties n_1, and subset $\mathcal{U}$. The adversary obtains $(\sigma_i)_{i\in\mathcal{U}}$ from the challenger and outputs $b' = 1$ if and only if $\mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{U}},\mathcal{U}) = \mu$. By the above argument, A wins with probability one. Moreover, condition (i) implies that $\mathcal{U}$ is unqualified, and thus A is valid. This finishes the proof. □

An evolving access structure that does not meet the properties (i) and (ii) in Theorem 5 (i.e., if $\mathcal{U} \notin \mathcal{A}_i$ for some i, then $\mathcal{U} \notin \mathcal{A}_j$ for all $j > i$) is called *rigid*. The definition below, which also appears in [23], formalizes this property.

Definition 20 (Rigid evolving access structure)**.** *A* rigid *evolving access structure $\mathcal{A} = \{\mathcal{A}_n\}$ is a monotone collection of subsets $\mathcal{A}_n \subseteq 2^{[n]}$ such that, for any $n \in \mathbb{N}$, it holds that $\mathcal{A}_n = \mathcal{A} \cap [n]$.*

Since in this paper we are not interested in evolving secret sharing schemes in which the dealer can update the shares, in what follows we only focus on rigid evolving access structures. Throughout the paper, we use the hat symbol $\hat{\mathcal{A}}$ to denote access structures that evolve while preserving rigidity.

4 Construction for General Access Structures

4.1 Exponential-time Construction

We present a construction of a secret sharing scheme for any rigid evolving access structure. The construction is based on any PRG with unbounded polynomial stretch (and thus only requires one-way functions). Unfortunately, for certain access structures, the running time of the sharing algorithm in our construction may be exponential, and thus we cannot prove computational privacy for all rigid evolving access structures. Nevertheless, we show that the construction runs in polynomial time for a fairly natural family of rigid evolving access structures, and in this case we can also prove computational privacy.

Construction 3

Let $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^*$ be a PRG. For a seed $\kappa \in \{0,1\}^\lambda$, we parse the output of $\mathsf{G}(\kappa)$ into blocks of size λ, and we denote with $\mathsf{G}(\kappa)[i]$ the i-th such block. Consider the following secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M} = \{0,1\}^\lambda$ for an arbitrary rigid evolving access structure $\hat{\mathcal{A}}$. We assume a lexicographic order $\xi : 2^{[n]} \to \mathbb{N}$ over the subsets in $\hat{\mathcal{A}}$, and that $\hat{\mathcal{A}}$ is in its minimal representation form.

Sharing: When the n-th party arrives, the dealer proceeds as follows:

- Sample $\kappa_n \leftarrow\!\!\$\ \{0,1\}^\lambda$.
- For each $\mathcal{I} \in \hat{\mathcal{A}}_n \setminus \hat{\mathcal{A}}_{n-1}$, let $\mathcal{I} = (i_1, \ldots, i_{t-1}, n)$ for some $t \geq 1$ and compute $\gamma_{\mathcal{I}} = \mu \bigoplus_{j=1}^{t-1} \mathsf{G}(\kappa_{i_j})[\xi(\mathcal{I})]$.
- Return $\sigma_n = (\kappa_n, (\gamma_{\mathcal{I}})_{\mathcal{I} \in \hat{\mathcal{A}}_n \setminus \hat{\mathcal{A}}_{n-1}})$ to the party.

Reconstruction: The reconstruction algorithm $\mathsf{SS.Recon}((\sigma_i)_{i \in \mathcal{I}}, \mathcal{I})$, given the shares $(\sigma_i)_{i \in \mathcal{I}}$ such that $\sigma_i = (\kappa_i, (\gamma_{\mathcal{I}})_{\mathcal{I} \in \hat{\mathcal{A}}_i \setminus \hat{\mathcal{A}}_{i-1}})$, and a minimal authorized subset $\mathcal{I} = \{i_1, \ldots, i_{t-1}, n\} \in \hat{\mathcal{A}}_n$, returns the same as $\gamma_{\mathcal{I}} \bigoplus_{j=1}^{t-1} \mathsf{G}(\kappa_{i_j})[\xi(\mathcal{I})]$, where $\gamma_{\mathcal{I}}$ is taken from the share σ_n of party n.

Recall that, by rigidity, every new authorized subset $\mathcal{I} \in \hat{\mathcal{A}}_n \setminus \hat{\mathcal{A}}_{n-1}$ must include the new party n. Correctness then follows by the fact that each of the party i_j in $\mathcal{I}$ knows the seed κ_{i_j}, and moreover can determine the correct index $\xi(\mathcal{I})$ given the lexicographic order ξ of the access structure $\hat{\mathcal{A}}$. However, the the sharing algorithm requires to generate a ciphertext $\gamma_{\mathcal{I}}$ for each minimal authorized subset $\mathcal{I} \in \hat{\mathcal{A}}_n \setminus \hat{\mathcal{A}}_{n-1}$ that party n completes, which can be exponential in n, and thus one cannot prove computational security of the above construction for arbitrary rigid evolving access structures.

4.2 Polynomial-time Instantiation

We note that if the number of authorized subsets that each new arriving party completes is $\mathsf{poly}(\lambda, n)$, then the sharing and reconstruction algorithms in Construction 3 always run in polynomial time. The size of each share is $\mathsf{poly}(\lambda, n)$, and becomes $\mathsf{poly}(\lambda)$ in case the number of added authorized subsets is independent of the number of parties currently in the system n. Moreover, in this case, the above secret sharing scheme is also computationally private. Below, we report the formal result.

Theorem 6. *Assuming* G *is secure (Definition 1), then the scheme* SS *described in Construction 3 is a computationally private secret sharing scheme over* $\mathcal{M} = \{0,1\}^\lambda$ *for every rigid evolving access structure* $\hat{\mathcal{A}}$ *such that, for each* $n \geq 1$*, it holds that* $|\hat{\mathcal{A}}_n| - |\hat{\mathcal{A}}_{n-1}| = \mathsf{poly}(\lambda, n)$.

Proof. Fix any rigid evolving access structure $\hat{\mathcal{A}} = \{\hat{\mathcal{A}}_n\}$ meeting the condition in the theorem statement. Let n and $\mathcal{U} \notin \hat{\mathcal{A}}_n$ be, respectively, the number of parties and the unqualified subset chosen by the adversary in the game defining computational privacy of SS. Denote by t the size of $\mathcal{U}$ and let $[n] \setminus \mathcal{U} = \{i_1, \ldots, i_{n-t}\}$. Consider the following hybrid experiments:

$\mathbf{H}_0(\lambda, b)$**:** This experiment is identical to $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$.

$\mathbf{H}_j(\lambda, b)$ **for** $j \in [n-t]$**:** Identical to $\mathbf{H}_{j-1}(\lambda, b)$ except that we change how the challenger computes the shares of the parties. In particular, for every party $k \in [n]$, and for every subset

$\mathcal{I}$ that is used during the computation of the share σ_k, and that contains the index i_j, the challenger replaces $\mathsf{G}(\kappa_{i_j})[\xi(\mathcal{I})]$ with a uniformly random string of length λ.

$\mathbf{H}_{n-t+j}(\lambda)$ for $j \in [n-t]$: Identical to $\mathbf{H}_{n-t+j-1}(\lambda, b)$ except that we change how the challenger computes the shares of the parties. In particular, for every party $k \in [n]$, and for every subset $\mathcal{I}$ that is used during the computation of the share σ_k, and that contains the index i_j, the challenger replaces $\gamma_{\mathcal{I}}$ with a uniformly random string of length λ.

Lemma 8. *For every $j \in [n-t]$, it holds that $\{\mathbf{H}_{j-1}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_j(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. The only difference between $\mathbf{H}_{j-1}(\lambda, b)$ and $\mathbf{H}_{j,b}(\lambda)$ is in the computation of the ciphertexts $\gamma_{\mathcal{I}}$ corresponding to authorized sets $\mathcal{I}$ that contain the index i_j. Since $i_j \notin \mathcal{U}$, the corresponding seed κ_{i_j} is unknown to the adversary. This allows to make a reduction to the security of the PRG G (Definition 1). The reduction is standard, so we omit it. □

Lemma 9. *For every $j \in [n-t]$, it holds that*

$$\{\mathbf{H}_{n-t+j-1}(\lambda, b)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_{n-t+j}(\lambda, b)\}_{\lambda\in\mathbb{N}}.$$

Proof. The lemma follows by simply observing that the distribution of ciphertexts $\gamma_{\mathcal{I}}$ corresponding to authorized sets $\mathcal{I}$ that contain the index i_j are now uniformly random. □

Lemma 10. $\{\mathbf{H}_{2n-2t}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_{2n-2t}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. The lemma follows by simply observing that the distribution of the shares $(\sigma_i)_{i\in\mathcal{U}}$ is now independent of the message. □

Theorem 7 now follows by combining the above lemmas. □

While the above construction is general, the shares are not succinct and additionally it fails to capture many important evolving access structures in which the number of added authorized subsets is super-polynomial in the number of parties (e.g., the dynamic threshold access structure). We will provide more efficient solutions for many of these access structures in the following section.

5 Constructions for Specific Access Structures

5.1 Dynamic Threshold Access Structure

We start with the so-called dynamic threshold evolving access structure [25]. This access structure is a generalization of the t-threshold evolving access structure, in which the authorized parties consist of all subsets of at least t parties, where $t = O(1)$ is fixed and independent of n. In the more general case of the dynamic threshold evolving access structure $\hat{\mathcal{A}}^{\mathsf{dthr}}$, instead, we have a sequence of thresholds $t_1 \leq t_2 \leq \cdots$, such that when there are n parties the qualified sets are those of size at least t_n; note that now, at least in general, the thresholds can depend on n. The condition that $t_n \geq t_{n-1}$ is necessary to ensure monotonicity, namely for the sequence of access structures to be a valid evolving access structure. Moreover, by definition, $\hat{\mathcal{A}}^{\mathsf{dthr}}$ is automatically rigid.

Below, we give a construction of secret sharing for $\hat{\mathcal{A}}^{\mathsf{dthr}}$ with message space $\{0,1\}^\lambda$ and share size $\lambda \cdot (n+1)$. The scheme is based on Shamir's secret sharing and on any standard PRG with unbounded polynomial stretch.

Construction 4

Let $\mathbb{GF}(2^\lambda)$ be a field of size 2^λ, and $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^*$ be a standard PRG with unbounded polynomial stretch. For a seed $\kappa \in \{0,1\}^\lambda$, we parse the output of $\mathsf{G}(\kappa)$ into blocks of size λ, and we denote with $\mathsf{G}(\kappa)[i]$ the i-th such block. Consider the following secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M} = \{0,1\}^\lambda$ for the dynamic threshold access structure $\hat{\mathcal{A}}^{\mathsf{dthr}}$ with sequence of thresholds $t_1 \leq t_2 \leq \cdots$.

Sharing: When the n-th party arrives, the dealer proceeds as follows:

- Sample a random polynomial f_n of degree $t_n - 1$ over $\mathbb{GF}(2^\lambda)[X]$, subject to $f_n(0) = \mu$.
- For every $i \in [n-1]$, compute $\gamma_i = f_n(i) \oplus \mathsf{G}(\kappa_i)[n-i]$.
  Finally, set the share of the n-th player to:

$$\sigma_n = (\kappa_n, (\gamma_i)_{i<n}, f_n(n))$$

  where $\kappa_n \leftarrow\!\!\$\ \{0,1\}^\lambda$.

Reconstruction: The reconstruction algorithm $\mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{I}}, \mathcal{I})$, given the shares $(\sigma_i)_{i\in\mathcal{I}} = ((\kappa_i, (\gamma_j)_{j<i}, y_i))_{i\in\mathcal{Q}}$, and a minimal authorized subset $\mathcal{I} = \{i_1, \ldots, i_{t_{n-1}}, n\} \in \hat{\mathcal{A}}_n^{\mathsf{dthr}}$, proceeds as follows:

- For every $j \in [t_n - 1]$, compute $y'_j = \gamma_{i_j} \oplus \mathsf{G}(\kappa_{i_j})[n - i_j]$ where $(\gamma_k)_{k<n}$ are the ciphertexts contained in σ_n.
- Use Lagrange interpolation over the points $((i_1, y'_1), \ldots, (i_{t_n-1}, y'_{t_n-1}), (n, y_n))$, yielding a polynomial $f' \in \mathbb{GF}(2^\lambda)[X]$, where $y_n = f_n(n)$ is the (plaintext) evaluation contained in σ_n.

Finally, output $f'(0) = \mu$.

Let $\mathcal{I} = \{i_1, \ldots, i_{t_n-1}, n\} \in \hat{\mathcal{A}}_n^{\mathsf{dthr}}$ be a minimal authorized subset, i.e., $\mathcal{I}$ is of size t_n. Correctness follows by observing that, for $j \in [n-1]$, each ciphertext γ_{i_j} contained in σ_n can be correctly decrypted by means of the seed κ_{i_j} contained in σ_{i_j}, thus yielding $y'_j = \gamma_{i_j} \oplus \mathsf{G}(\kappa_{i_j})[n - i_j] = f_n(i_j)$. Hence, the authorized parties can retrieve t_n evaluations (as $f_n(n) = y_n$ is already contained in σ_n) of the polynomial f_n of degree $t_n - 1$. Thus, by correctness of Lagrange interpolation, the parties in $\mathcal{I}$ can correctly reconstruct $f' = f_n$ and, in turn, recover the correct message $f'(n) = f_n(0) = \mu$. As for security, we prove the following result.

Theorem 7. *Assuming* G *is secure (Definition 1), the scheme* SS *described in Construction 4 is a computationally private secret sharing scheme for the dynamic threshold evolving access structures* $\hat{\mathcal{A}}^{\mathsf{dthr}}$.

Proof. Let n and $\mathcal{U} \notin \hat{\mathcal{A}}_n^{\mathsf{dthr}}$ be, respectively, the number of parties and the unqualified subset chosen by the adversary in the game defining computational privacy of SS. Also, let $t_1 \leq t_2 \leq \cdots \leq t_n$ be the thresholds defining $\hat{\mathcal{A}}_n^{\mathsf{dthr}}$. Consider the following hybrid experiments:

$\mathbf{H}_0(\lambda, b)$**:** This experiment is identical to $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$.

$\mathbf{H}_j(\lambda, b)$ **for** $j \in [n] \setminus \mathcal{U}$**:** Identical to $\mathbf{H}_{j-1}(\lambda, b)$, except that we change how the challenger computes the shares of the parties. In particular, for every party $k \in [n]$, the challenger (during

the computation of the σ_k) lets $\gamma_j = f_k(j) \oplus \rho$ (recall that γ_j is part of the share σ_k) where $\rho \leftarrow\!\!\$\ \{0,1\}^\lambda$.

$\mathbf{H}_{|[n]\setminus\mathcal{U}|+j}(\lambda, b)$ for $j \in [n] \setminus \mathcal{U}$: Identical to $\mathbf{H}_{|[n]\setminus\mathcal{U}|+j-1}(\lambda, b)$, except that we change how the challenger computes the shares of the parties. In particular, for every party $k \in [n]$, the challenger (during the computation of σ_k) samples γ_j at random from $\{0,1\}^\lambda$ (recall that γ_j is part of the share σ_k).

Lemma 11. *For every $j \in [n] \setminus \mathcal{U}$, and for every $b \in \{0,1\}$, it holds that $\{\mathbf{H}_{j-1}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_j(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. The only difference between $\mathbf{H}_{j-1}(\lambda, b)$ and $\mathbf{H}_j(\lambda, b)$ is in the computation of the ciphertexts $\{\gamma_j\}$ contained in each of the shares $(\sigma_1, \ldots \sigma_n)$. Since $j \notin \mathcal{U}$, we have that the seed κ_j of the j-th party is never revealed to the adversary. Hence, the lemma follows by the security of the PRG G (Definition 1). □

Lemma 12. *For every $j \in [n]\setminus\mathcal{U}$, and for every $b \in \{0,1\}$, it holds that $\{\mathbf{H}_{|[n]\setminus\mathcal{U}|+j-1}(\lambda, b)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_{|[n]\setminus\mathcal{U}|+j}(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. The lemma follows by simply observing that, for every party $k \in [n]$, each ciphertext γ_j (contained in the share σ_k) for $j \notin \mathcal{U}$ is a one-time pad encryption which, in turn, is identically distributed to a random string in $\{0,1\}^\lambda$. □

Lemma 13. $\{\mathbf{H}_{2\cdot|[n]\setminus\mathcal{U}|}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_{2\cdot|[n]\setminus\mathcal{U}|}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. By the validity of the adversary, we have that $|\mathcal{U} \cap [k]| < t_k$ for every $k \in [n]$. Otherwise, we would have that $\mathcal{U} \in \hat{\mathcal{A}}^{\mathsf{dthr}}_k$ by definition of the dynamic threshold access structure. It easy to see that the above condition implies that, for every $k \in [n]$, in experiment $\mathbf{H}_{2\cdot|[n]\setminus\mathcal{U}|}(\lambda, b)$ (for $b \in \{0,1\}$) the challenger evaluates (and uses) at most $|\mathcal{U} \cap [k]| < t_k$ evaluations of the polynomial f_k sampled at round k. Hence, we can use the security of Shamir's secret sharing in order to conclude the proof since, for each polynomial f_k (recall that f_k is of degree $t_k - 1$), the point $f_k(0)$ is information-theoretically hidden given at most $t_k - 1$ evaluations of the polynomial. This concludes the proof. □

Theorem 7 follows by combining the above lemmas. We note that the exact same proof allows to show that Construction 4 is a computationally private secret sharing scheme for the evolving flexible dynamic threshold access structure $\hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}}$, in which the thresholds are arbitrary. The reason for this is that the validity condition of the adversary for this access structure still implies $|\mathcal{U} \cap [k]| < t_k$ for every $k \in [n]$, which allows for Lemma 13 to go through. □

Remark 2 (Static threshold)**.** *A special case of the dynamic threshold access structure is the (static) t-threshold access structure $\hat{\mathcal{A}}^{\mathsf{thr}}_t$, i.e., the threshold t is fixed and independent of n (i.e., $t_1 = t_2 = \cdots = t = O(1)$). Of course, Construction 4 yields a secret sharing scheme for $\hat{\mathcal{A}}^{\mathsf{thr}}_t$, with share size $\lambda \cdot (n+1)$.*[11] *However, the share size can be improved with a direct construction based on Shamir's secret sharing scheme: The dealer samples a single random polynomial f over $\mathbb{GF}(2^\lambda)$, of degree $t-1$, and subject to $f(0) = \mu$. Then, for each $n \geq 1$, the share of party n is simply $\sigma_n = f(n)$. The share size is λ. Note that this works because, in the computational setting, the number of parties is upper bounded by an unknown polynomial, but the field is of exponential size.*

[11] Note that a secret sharing scheme for $\hat{\mathcal{A}}^{\mathsf{thr}}_t$ can also be obtained as a special case of our construction for rigid evolving DNF formulas access structures (see Theorem 11), by taking $n_1 = t$, $n_g = 1$ for every $g \geq 2$, and $m_g = \binom{n}{t}$ for every $g \geq 1$. However, the latter yields an even worse share size of $\mathsf{poly}(\lambda, n)$.

Remark 3 (Flexible dynamic threshold)**.** *Consider the evolving* flexible *dynamic threshold access structure* $\hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}} = \{t_n\}$*, in which the* new *authorized subsets when the* n*-th party arrives (i.e., the subsets in* $\hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}}_n \setminus \hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}}_{n-1}$*) are all the subsets of at least* t_n *players* that always include n*. We note that this access structure is still monotone, even if we remove the condition that* $t_1 \leq \cdots \leq t_n$*; in fact, when the latter condition is added, the access structure* $\hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}}$ *collapses to* $\hat{\mathcal{A}}^{\mathsf{dthr}}$*. One can show that Construction 4 yields a computationally private secret sharing scheme for* $\hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}}$*, with exactly the same parameters. See the proof of Theorem 7 for more on this point.*

Corollary 1. *Assuming OWFs, there exists a computationally private secret sharing scheme over* $\mathcal{M} = \{0,1\}^\lambda$ *for the evolving flexible dynamic threshold access structure* $\hat{\mathcal{A}}^{\mathsf{flex\text{-}dthr}}$*, where the share size of party* $n \geq 1$ *is* $\lambda \cdot (n+1)$*.*

5.2 Graphs

We start by recalling the concept of secret sharing schemes for graph access structures. Given an undirected graph $G = (\mathcal{V}, \mathcal{E})$, we view the parties as the vertexes in $\mathcal{V}$ and authorized sets are those sets that contain vertexes such that there is at least a pair of vertexes corresponding to an edge in the graph. Namely, the access structure is specified by a function $f_G : \{0,1\}^n \to \{0,1\}$, where $n = |\mathcal{V}|$, and $f_G(x) = 1$ if and only if there exist indexes $i, j \in [n]$ such that $x_i = x_j = 1$ and $(i,j) \in \mathcal{E}$.

We can generalize the above access structure to the evolving setting as follows. W.l.o.g., parties are added to the graph in an online manner; every new player is added to the vertex set $\mathcal{V}$ and can be connected via one or more edges to the previous nodes in the graph. However, no new edges can be added between old nodes in the set $\mathcal{V}$, i.e. we only consider the rigid evolving graphs access structure, which we denote by $\hat{\mathcal{A}}^{\mathsf{graph}} = \{\mathcal{E}_n\}$, where $G_n = (\mathcal{V}_n, \mathcal{E}_n)$ is the graph when there are n parties. By Theorem 5, the above limitation is inherent as the evolving secret sharing scheme we describe below does not require to update the shares of old players.

Following [7], we focus on the simpler case where the graph $G_n = (\mathcal{V}_n, \mathcal{E}_n)$ is bipartite, namely the vertex set $\mathcal{V}_n$ consists of two sets $(\mathcal{V}^{(0)}_n, \mathcal{V}^{(1)}_n)$, and the edge set $\mathcal{E}_n$ is of the form $\mathcal{E}_n \subseteq \mathcal{V}^{(0)}_n \times \mathcal{V}^{(1)}_n$. This naturally extends to the evolving setting as well (keeping the rigidity condition), and we write $\hat{\mathcal{A}}^{\mathsf{2graph}} = \{\mathcal{E}_n\}$ to denote the rigid evolving bipartite graphs access structure. We will later show that secret sharing for $\hat{\mathcal{A}}^{\mathsf{2graph}}$ implies secret sharing for $\hat{\mathcal{A}}^{\mathsf{graph}}$.

Construction 5

Let $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ be a pPRG with unbounded polynomial stretch. Consider the following secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M} = \{0,1\}$ for the rigid evolving bipartite graphs access structure $\hat{\mathcal{A}}^{\mathsf{2graph}}$.

Sharing: At the onset, the dealer computes $(\mathsf{mpk}_b, \mathsf{msk}_b) \leftarrow\!\!\$\ \mathsf{pPRG.Setup}(1^\lambda)$ for every $b \in \{0,1\}$. When the n-th party arrives, the dealer proceeds as follows:

- Let $v_n \in \mathcal{V}^{(b)}_n$ (for some $b \in \{0,1\}$) be the node in the graph corresponding to the n-th party, where $G_n = ((\mathcal{V}^{(0)}_n, \mathcal{V}^{(1)}_n), \mathcal{E}_n)$ is the graph specified in $\hat{\mathcal{A}}^{\mathsf{2graph}}_n$.
- Run $\alpha_{\{n\}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}_b, \mathsf{msk}_b, \{n\})$ and $y_n = \mathsf{pPRG.Eval}(\mathsf{mpk}_b, \alpha_{\{n\}}, \{n\})$, and let $\gamma_n = \mu \oplus y_n$.
- Let $\mathcal{T}_n = \{j : (v_n, v_j) \in \mathcal{E}_n\}$ be the set of indexes corresponding

to the neighbors of the node $v_n \in \mathcal{V}_n^{(b)}$ in the graph. Let $\alpha_{\mathcal{T}_n} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}_{1-b}, \mathsf{msk}_{1-b}, \mathcal{T}_n)$.

- Return $\sigma_n = (\mathsf{mpk}_{1-b}, \gamma_n, \alpha_{\mathcal{T}_n})$.

Reconstruction: The reconstruction algorithm $\mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{I}}, \mathcal{I})$, given the shares $(\sigma_i)_{i\in\mathcal{I}} = ((\mathsf{mpk}, \gamma_i, \alpha_{\mathcal{T}_i}))_{i\in\mathcal{I}}$ (for some $b \in \{0,1\}$), and a minimal authorized subset $\mathcal{I} = \{v_{i_0}, v_{i_1}\}$ such that $(v_{i_0}, v_{i_1}) \in \mathcal{E}_n = \hat{\mathcal{A}}_n^{\mathsf{2graph}}$, proceeds as follows:

- Without loss of generality, assume that $v_{i_0} \in \mathcal{V}_n^{(b)}$ and $v_{i_1} \in \mathcal{V}_n^{(1-b)}$ (otherwise, simply swap v_{i_0} and v_{i_1}). Hence, $\sigma_0 = (\mathsf{mpk}_{1-b}, \gamma_{i_0}, \alpha_{\mathcal{T}_{i_0}})$ and $\sigma_1 = (\mathsf{mpk}_b, \gamma_{i_1}, \alpha_{\mathcal{T}_{i_1}})$
- If $i_0 < i_1$, compute $y' = \mathsf{pPRG.Eval}(\mathsf{mpk}_b, \alpha_{\mathcal{T}_{i_1}}, \mathcal{T}_{i_1})$ and $\mu = \gamma_{i_0} \oplus y'_k$, where y'_k is the k-th bit of y' such that the k-th index $i_k \in \mathcal{T}_{i_1}$ is equal to i_0.
- Otherwise, if $i_1 < i_0$, compute $y' = \mathsf{pPRG.Eval}(\mathsf{mpk}_{1-b}, \alpha_{\mathcal{T}_{i_0}}, \mathcal{T}_{i_0})$ and $\mu = \gamma_{i_1} \oplus y'_k$, where y'_k is the k-th bit of y' such that the k-th index $i_k \in \mathcal{T}_{i_0}$ is equal to i_1.
- Output μ.

Correctness follows readily from the correctness property of the underlying projective pPRG. More in details, fix two vertexes v_{i_1} and v_{i_0} such that $(v_{i_0}, v_{i_1}) \in \mathcal{E}_n$ and $v_{i_j} \in \mathcal{V}_n^{(j)}$ for $j \in \{0,1\}$. Assume $i_0 < i_1$ (the case $i_1 < i_0$ follows by using a symmetrical argument). Then, we have that $\gamma_{i_0} = \mu \oplus y_{i_0}$ (which is part of σ_{i_0}), where $y_{i_0} = \mathsf{pPRG.Eval}(\mathsf{mpk}_0, \alpha_{i_0}, \{i_0\})$. Moreover, by definition, we have that $\alpha_{\mathcal{T}_{i_1}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}_0, \mathsf{msk}_0, \mathcal{T}_{i_1})$ (which is part of σ_{i_1}), where $i_0 \in \mathcal{T}_{i_1}$. By correctness of the projective PRG, we conclude that the output $\mu = \gamma_{i_0} \oplus y'_k$ is correct, since y'_k is the k-th bit of $y' = \mathsf{pPRG.Eval}(\mathsf{mpk}_0, \alpha_{\mathcal{T}_{i_1}}, \mathcal{T}_{i_1})$ such that $i_0 = i_k \in \mathcal{T}_{i_1}$ (i.e., $y'_k = y_{i_0}$).

As for security, we prove the following theorem.

Theorem 8. *Assuming* pPRG *is robust (Definition 14), then the scheme* SS *described in Construction 5 is a computationally private secret sharing scheme over* $\mathcal{M} = \{0,1\}$ *for the the rigid evolving bipartite graphs access structure* $\hat{\mathcal{A}}^{\mathsf{2graph}}$.

Proof. Let n and $\mathcal{U} \notin \hat{\mathcal{A}}_n^{\mathsf{2graph}}$ be, respectively, the number of parties and the unqualified subset (i.e., vertexes that does not have an edge between them) chosen by the adversary in the game defining computational privacy of SS. Also, let $G_n = ((\mathcal{V}_n^{(0)}, \mathcal{V}_n^{(1)}), \mathcal{E}_n)$ be the graph which defines $\hat{\mathcal{A}}_n^{\mathsf{2graph}}$. Consider the following hybrid experiments:

$\mathbf{H}_0(\lambda, b)$: This experiment is identical to $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$.

$\mathbf{H}_1(\lambda, b)$ Identical to $\mathbf{H}_0(\lambda, b)$, except that we change how the challenger computes the shares of the parties. In particular, for every party $k \in \mathcal{U}$ such that $v_k \in \mathcal{V}_n^{(0)}$, the challenger (during the computation of σ_k) lets $\gamma_j = \mu \oplus \rho$ (recall that γ_j is part of σ_k), where $\rho \leftarrow\!\!\$\ \{0,1\}$.

$\mathbf{H}_2(\lambda, b)$: Identical to $\mathbf{H}_1(\lambda, b)$, except that we change how the challenger computes the shares of the parties. In particular, for every party $k \in \mathcal{U}$ such that $v_k \in \mathcal{V}_n^{(1)}$, the challenger (during the computation of σ_k) lets $\gamma_j = \mu \oplus \rho$ (recall that γ_j is part of σ_k), where $\rho \leftarrow\!\!\$\ \{0,1\}$.

Lemma 14. *For every* $b \in \{0,1\}$, *we have* $\{\mathbf{H}_0(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$.

Proof. By the validity of the adversary, we have that either $i \notin \mathcal{U}$ or $j \notin \mathcal{U}$ for every $(v_i, v_j) \in \mathcal{E}_n$. Hence, for each share σ_i such that $i \in \mathcal{U}$ (i.e., the share σ_i is obtained by the adversary)

and $v_i \in \mathcal{V}_n^{(1)}$, the corresponding projective key $\alpha_{\mathcal{T}_i}$ associated to set $\mathcal{T}_i$ satisfies the following invariant: $\forall k \in \mathcal{U}$ such that $k \in \mathcal{V}_n^{(0)}$ we have $k \notin \mathcal{T}_i$ (otherwise the adversary is not valid). Hence, by using a hybrid argument (over $k \in \mathcal{U}$ such that $v_k \in \mathcal{V}_n^{(0)}$) and the security of pPRG, we conclude that γ_k (as defined in $\mathbf{H}_0(\lambda, b)$) is computationally indistinguishable from $\mu \oplus \rho$ where $\rho \leftarrow\!\!\$\ \{0,1\}$ (as defined in $\mathbf{H}_1(\lambda, b)$), for every $k \in \mathcal{U}$ such that $v_k \in \mathcal{V}_n^{(0)}$. □

Lemma 15. *For every $b \in \{0,1\}$, we have $\{\{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_2(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. The lemma follows by using an analogous argument to that used in the proof of Lemma 14. □

Lemma 16. $\{\mathbf{H}_2(\lambda, 0)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_2(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. The lemma follows by observing that:

- The only values (obtained by the adversary) that depend on μ_0 and μ_1 are the ciphertexts $(\gamma_i)_{i\in\mathcal{U}}$ contained in the shares $(\sigma_i)_{i\in\mathcal{U}}$.
- Each γ_i for $i \in \mathcal{U}$ is a one-time pad encryption of the message.

Thus, the experiments $\mathbf{H}_2(\lambda, 0)$ and $\mathbf{H}_2(\lambda, 1)$ are identically distributed. □

Theorem 8 follows by combining the above lemmas. □

For each $n \geq 1$, the share σ_n in Construction 5 consists of a one-bit ciphertext γ_n, a projective key $\alpha_{\mathcal{T}_n}$, and the master public key mpk of the projective PRG. Moreover, while the construction only deals with message space $\mathcal{M} = \{0,1\}$, it is immediate to obtain a scheme for $\mathcal{M} = \{0,1\}^\lambda$ by repeating the construction λ times in parallel. Hence, by invoking Theorem 2, we obtain:

Corollary 2. *Under the RSA assumption, or assuming iO and SSB hash functions, there exists a computationally private secret sharing scheme over $\mathcal{M} = \{0,1\}^\lambda$ for the the rigid evolving bipartite graphs access structure $\hat{\mathcal{A}}^{\mathsf{2graph}}$, where the share size of party $n \geq 1$ is $\mathsf{poly}(\lambda)$.*

Arbitrary graphs. As observed in [7], in the non-evolving setting, secret sharing schemes for bipartite graph access structures imply ones for arbitrary graph access structures. In more details, given a graph $G = (\mathcal{V}, \mathcal{E})$ one can construct a bipartite graph H by taking two copies of each vertex, and for every $(u, v) \in \mathcal{E}$ connect the first copy of u to the second copy of v; the share of each party in $\mathcal{V}$ consists of the shares of the corresponding two copies of this vertex in H. The above transformation clearly preserves rigidity, and thus readily adapts to the setting of rigid evolving graphs access structures. Thus, by leveraging Corollary 2, we obtain:

Corollary 3. *Under the RSA assumption, or assuming iO and SSB hash functions, there exists a computationally private secret sharing scheme over $\mathcal{M} = \{0,1\}^\lambda$ for the rigid evolving graphs access structure $\hat{\mathcal{A}}^{\mathsf{graph}}$, where the share size of party $n \geq 1$ is $\mathsf{poly}(\lambda)$.*

5.3 Monotone Circuits

We now turn to the case of access structures represented as circuits $C : \{0,1\}^n \to \{0,1\}$ with AND and OR gates of unbounded fan-in, which we refer to as an AND-OR circuit. We denote by $x = (x_1, \ldots, x_n)$ the input to the circuit, where $x_i = 1$ means that the i-th player is part of the reconstruction. Following [1], we make some conventions on the structure of the circuit.

- We assume w.l.o.g. that all the outgoing wires of an OR gate are connected as incoming wires to AND gates, and viceversa; if this is not the case and, say, an OR gate has an outgoing wire that enters another OR gate, we can duplicate all the input wires of the first OR gate and connect them directly to the second OR gate. (The same can be done with AND gates.)

- We assume, for simplicity, that each input wire is only[12] connected to an OR gate with fan-in 1; this can be achieved by adding an OR gate with fan-in 1 for every input x_i that goes into an AND gate with fan-in ≥ 2, and by adding both an OR gate with fan-in 1 followed by an AND gate with fan-in 1 for every input x_i that goes into an OR gate with fan-in ≥ 2 (at the cost of increasing the number of gates by at most $2n$).

- We assume that gates are numbered from 1 to m according to some topological order and that the first n gates correspond to the inputs $x_1, \ldots, x_n$. We write $i \to j$ when an output of the i-th gate is being fed to the j-th gate as an input.

As explained in the introduction, we can generalize the above to the evolving setting as follows. The rigid evolving monotone circuits access structure $\hat{\mathcal{A}}^{\mathsf{ckts}} = \{\hat{\varphi}_g\}_{g \geq 1}$ consists of a sequence of monotone formulas $\hat{\varphi}_g$ (with $g = \mathsf{poly}(\lambda)$) defined as follows:

$$\hat{\varphi}_g(x) = \bigvee_g \hat{C}_g(x_1, \ldots, x_n), \tag{1}$$

where $\hat{C}_g : \{0,1\}^n \to \{0,1\}$ is an arbitrary AND-OR circuit such that $\hat{C}_g(x_1, \ldots, x_{n-n_g}, 0, \ldots, 0) = 0$, and $n = \sum_g n_g$. Furthermore, without loss of generality, we will assume the output gates in $\hat{C}_g$ are all AND gates, as if an OR gate is an output gate we can remove it and connect its output to the final OR gate in the above formula (i.e., the operator $\bigvee_g$ in Equation (1)).

Based on the above formalization, we propose an evolving secret sharing scheme for $\hat{\mathcal{A}}^{\mathsf{ckts}}$ based on projective PRGs with *bounded* polynomial stretch (and standard PRGs with unbounded polynomial stretch).

Construction 6

Let $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ be a block projective PRG with bounded polynomial stretch, and $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^*$ be a standard PRG with unbounded polynomial stretch. For a seed $\kappa \in \{0,1\}^\lambda$, we parse the output of $\mathsf{G}(\kappa)$ into blocks of size equal to the size of a projective key produced by pPRG, and denote with $\mathsf{G}(\kappa)[i]$ the i-th such block. Consider the following secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M} = \{0,1\}^\lambda$ for the rigid evolving monotone circuits access structure $\hat{\mathcal{A}}^{\mathsf{ckts}}$.

Sharing: When the generation $g \geq 1$ begins, the dealer proceeds as follows:

- Let $\hat{C}_g$ be the AND-OR circuit corresponding to the arrival of the g-th generation (see Equation (1)). Let $m_g = m_g^\wedge + m_g^\vee$ be the number of gates in $\hat{C}_g$, where $m_g^\wedge$ and $m_g^\vee$ are, respectively, the number of AND and OR gates (including the input and output gates).
- Compute $(\mathsf{mpk}_g, \mathsf{msk}_g) \leftarrow\!\!\$\ \mathsf{pPRG.Setup}(1^\lambda, 1^{m_g^\vee})$.

[12] This assumption is slightly different from[1], where one adds an OR gate with fan-in 1 only for the input wires that go into AND gates; the modification is needed in order to obtain a construction in the evolving setting.

- For each $i \in [m_g^\vee]$, associate to the i-th OR gate in $\hat{C}_g$ a key $\kappa_i^{(g)}$ determined as follows:
  - If the gate is an input OR gate, set $\kappa_i^{(g)} = \mathsf{G}(\kappa_i^*)[1]$ where $\kappa_i^* \leftarrow\!\!\$\ \{0,1\}^\lambda$ for each $i \in [n] \setminus [n - n_g]$, and $\kappa_i^{(g)} = \mathsf{G}(\kappa_i^*)[g - g_i + 1]$ for each $i \in [n - n_g]$, where g_i is the generation corresponding to the arrival of party $i \in [n - n_g]$. (Observe that, for every $i \in [n - n_g]$, the key κ_i^* corresponds to the PRG seed contained in the shares of parties of previous generations.)
  - If the gate is a non-input OR gate, set

$$\kappa_i^{(g)} = y_i^{(g)} = \mathsf{pPRG.Eval}(\mathsf{mpk}_g, \alpha_{\{i\}}^{(g)}, \{i\}),$$

    where $\alpha_{\{i\}}^{(g)} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}_g, \mathsf{msk}_g, \{i\})$.
- For each $i \in [m_g^\wedge]$, associate to the i-th AND gate in $\hat{C}_g$ a key $\kappa_i^{(g)}$ determined as follows:
  - If the i-th gate is a non-output AND gate, set

$$\kappa_i^{(g)} = \alpha_{\mathcal{T}_i^{(g)}} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}_g, \mathsf{msk}_g, \mathcal{T}_i^{(g)}),$$

    where $\mathcal{T}_i^{(g)} = \{j : i \to j \text{ in } \hat{C}_g\}$ consists of all out-neighbor of the i-th AND gate in $\hat{C}_g$.
  - If the i-th gate is an output AND gate, set $\kappa_i^{(g)} = \mu$.
- For each $i \in [m_g^\wedge]$, associate to the i-th AND gate in $\hat{C}_g$ (including the output gates) the ciphertext $\gamma_i^{(g)} = \kappa_i^{(g)} \oplus \rho_i^{(g)}$, which is viewed as an encryption of $\kappa_i^{(g)}$ under the mask $\rho_i^{(g)} = \bigoplus_{j:j\to i} \mathsf{G}(\kappa_j^{(g)})[i]$. (For each of the output AND gates, we only take the first λ bits of the corresponding PRG blocks.)
- Finally, the share of party $i \in [n] \setminus [n - n_g]$ (i.e, the share of the new parties belonging to the g-th generation) is defined to be $\sigma_i = (\mathsf{mpk}_g, \kappa_i^*, (\gamma_j^{(g)})_{j\in[m_g^\wedge]})$.

Reconstruction: Let $x \in \{0,1\}^n$ be the input that corresponds to the parties that want to reconstruct the message. For each generation, we traverse the circuit $\hat{C}_g$ from the inputs to the outputs, and recover the key $\kappa_i^{(g)}$ associated to each gate i that is satisfied by x (i.e., the gate is evaluated to 1 under the assignment x) in $\hat{C}_g$. In case the latter allows to obtain the key associated to any of the output AND gates in $\hat{C}_g$, output that value as the reconstructed message.

Correctness can be seen as follows. Clearly, for any qualified subset $\mathcal{I}$ corresponding to a satisfying assignment $x_\mathcal{I}$ for $C(x)$, there exists some $g \geq 1$ such that at least one output AND gate in $\hat{C}_g$ is satisfied by $x_\mathcal{I}$. Furthermore, the parties in $\mathcal{I}$ can recover the key $\kappa_i^{(g)}$ associated to each gate i that is satisfied by $x_\mathcal{I}$ in $\hat{C}_g$, and thus can correctly recover the message. Indeed:

- For an input OR gate, the key $\kappa_i^{(g)}$ is given as part of the shares of the parties corresponding to the assignment x.
- For a non-input OR gate, the key $\kappa_i^{(g)} = y_i^{(g)}$ can be recovered based on the key $\kappa_j^{(g)}$ of the first gate j that is satisfied, and whose outgoing wire enters i, i.e. $j \to i$ in $\hat{C}_g$. Indeed, j

is an AND gate, and its key $\kappa_j^{(g)}$, which was already recovered, consists of a projective key $\alpha_{\mathcal{T}_j^{(g)}}$ for a set $\mathcal{T}_j^{(g)}$ that contains i.

- For an AND gate, the key $\kappa_i^{(g)}$ can be recovered by XOR-ing the ciphertext $\gamma_i^{(g)}$ with the mask $\rho_i^{(g)}$. This mask can be computed based on all the keys $\{\kappa_j^{(g)} : j \to i \text{ in } \hat{C}_g\}$ that were already recovered (since $j < i$ and since all the gates $j : j \to i$ must be satisfied by $x_{\mathcal{I}}$).

Turning to security, we establish the following result.

Theorem 9. *Assuming* pPRG *is robust (Definition 14) and* G *is secure (Definition 1), the scheme* SS *described in Construction 6 is a computationally private secret sharing scheme over* $\mathcal{M} = \{0,1\}^\lambda$ *for the rigid evolving monotone circuits access structure* $\hat{\mathcal{A}}^{\mathsf{ckts}}$.

Proof. Let n and $\mathcal{U}$ be, respectively, the number of parties and the unqualified subset chosen by the adversary in the game defining computational privacy of SS. Denote by $t = |\mathcal{U}|$ the total number of corrupted parties, and let $[n] \setminus \mathcal{U} = \{i_1, \ldots, i_{n-t}\}$. By validity of the adversary, the unqualified subset $\mathcal{U}$ is associated to an input $x_{\mathcal{U}} \in \{0,1\}^n$ such that $\bigvee_g \hat{C}_g(x_{\mathcal{U}}) = 0$, where $\hat{C}_g$ is the AND-OR circuit corresponding to the g-th generation. We denote by $\bar{g}$ the last generation. Consider the following hybrid experiments:

$\mathbf{H}_0(\lambda, b)$**:** This experiment is identical to $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$.

$\mathbf{H}_1(\lambda, b)$**:** We change how the challenger computes the shares of the parties. In particular, for every generation $g \geq 1$, we replace the key $\kappa_i^{(g)}$ associated to every input OR gate corresponding to an honest party, with a uniformly random string of the appropriate length.

$\mathbf{H}_2(\lambda, b)$**:** We change how the challenger computes the shares of the parties. In particular, for every generation $g \geq 1$:

- We replace the key $\kappa_i^{(g)}$ associated to an unsatisfied non-input OR gate with a uniformly random key of the appropriate length;
- We replace the ciphertext $\gamma_i^{(g)}$ associated to an unsatisfied AND gate (including the output AND gates) with a uniformly random ciphertext of the appropriate length.

Lemma 17. *For every* $b \in \{0,1\}$*, it holds that* $\{\mathbf{H}_0(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$.

Proof. The proof uses the hybrid argument. In particular, for every $j \in [0, n-t]$, let $\mathbf{H}_{1,j}(\lambda, b)$ be the experiment where, for every generation $g \geq 1$, we replace the key $\kappa_i^{(g)}$ associated to each of the input OR gates that correspond to the first j honest players with a uniformly random key $\kappa_i^{(g)}$, whereas the keys $\kappa_i^{(g)}$ associated to each of the input OR gates that correspond to the remaining players are defined as in $\mathbf{H}_1(\lambda, b)$. Clearly, $\{\mathbf{H}_{1,0}(\lambda, b)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_0(\lambda, b)\}_{\lambda\in\mathbb{N}}$ and $\{\mathbf{H}_{1,n-t}\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$, and thus, in order to prove the lemma, it suffices to show that, for all $j \in [n-t]$, we have $\{\mathbf{H}_{1,j}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{1,j-1}(\lambda, b)\}_{\lambda\in\mathbb{N}}$.

The latter statement follows by security of the PRG G. Indeed, the only difference between $\mathbf{H}_{1,j-1}(\lambda, b)$ and $\mathbf{H}_{1,j}(\lambda, b)$ is that, for every $g \geq 1$, all of the keys $\kappa_i^{(g)}$ associated to the input OR gates that correspond to the j-th honest player are either equal to $\mathsf{G}(\kappa^*_{i_j})[g - g_{i_j}]$ (where the seed of the PRG is unknown to the adversary) or uniformly random. The reduction is standard, and thus we omit it. □

Lemma 18. *For every $b \in \{0,1\}$, it holds that $\{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_2(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. The proof uses the hybrid argument. In particular, for every $g \leq \bar{g}$ and $h \leq m_g$, let $\mathbf{H}_{2,g,h}(\lambda, b)$ be the experiment specified below:

- For the first $g-1$ generations, the shares of the players are computed as in $\mathbf{H}_2(\lambda, b)$.
- For the generation g, the shares of the players are computed as follows:
  - For all $i \leq h$, if the i-th gate is an unsatisfied AND gate, sample the ciphertext $\gamma_i^{(g)}$ uniformly at random; if the i-th gate is an unsatisfied OR gate, sample the corresponding key $\kappa_i^{(g)}$ uniformly at random.
  - For all $i > h$, On the other hand, the keys and ciphertexts associated to gates $i > h$ are determined as in $\mathbf{H}_1(\lambda, b)$.
- For the last $\bar{g} - g$ generations, the shares of the players are computed as in $\mathbf{H}_1(\lambda, b)$.

Clearly, $\{\mathbf{H}_{2,0,0}(\lambda, b)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$ and $\{\mathbf{H}_{2,\bar{g},m_{\bar{g}}}\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_2(\lambda, b)\}_{\lambda\in\mathbb{N}}$, and thus, in order to prove the lemma, it suffices to show that, for all $g \in [\bar{g}]$ and all $h \in [n_g]$, we have $\{\mathbf{H}_{2,g,h}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{2,g,h-1}(\lambda, b)\}_{\lambda\in\mathbb{N}}$.

Observe that the two hybrids differ only if the h-th gate is an unsatisfied gate which is either an AND gate or a non-input OR gate. We deal with these two cases separately.

Case 1: the h-th gate is an AND gate. In this case, at least one of the incoming wires j must be connected to an unsatisfied OR gate $j < h$. The key $\kappa_j^{(g)}$ associated to this gate is chosen at random in both hybrids, so we can use any distinguisher A between the two hybrids in order to break the security of G as follows:

- The reduction is given[13] $m_g^{\wedge}$ blocks of size equal to the dimension of a projective key producede by pPRG, which we denote by $z = (z_1, \ldots, z_{m_g^{\wedge}})$, where z_i is uniform for all $i \leq h-1$ and $z_i = \mathsf{G}(\kappa_j^{(g)})[i]$ for all $i \geq h+1$, and where z_h is either uniform or equal to $z_h = \mathsf{G}(\kappa_j^{(g)})[h]$.
- For the first $g-1$ generations, the reduction computes the shares of the players exactly as defined in $\mathbf{H}_2(\lambda, b)$.
- For the generation g, the reduction determines the shares of the parties as defined in $\mathbf{H}_1(\lambda, b)$, except that it replaces the key $\kappa_i^{(g)}$ with the challenge block z_i.
- For the last $\bar{g} - g$ generations, the reduction computes the shares of the players exactly as defined in $\mathbf{H}_1(\lambda, b)$.
- Finally, output whatever A outputs.

Depending on the distribution of the challenge z, the reduction perfectly simulates the view of the adversary in either $\mathbf{H}_{2,g,h}(\lambda, b)$ or in $\mathbf{H}_{2,g,h-1}(\lambda, b)$. The lemma follows.

[13] This property follows by security of G by a standard hybrid argument.

Case 2: the h-th gate is a non-input OR gate. In this case, all the incoming wires must be connected to unsatisfied gates. We show how to turn any distinguisher A between the two hybrids into an adversary attacking the robustness property of the projective PRG pPRG as follows:

- The reduction initializes the robustness game with parameters $1^{m_g^\vee}$, and asks the challenger to obtain the projective keys associated to all sets $\mathcal{T}_i^{(g)}$ that are used by the dealer in the computation of the shares for the g-th generation, except for the sets that contain the index h. Let $\bar{\mathcal{T}}$ be the union of those sets.
- For the first $g-1$ generations, the reduction computes the shares of the players exactly as defined in $\mathbf{H}_2(\lambda, b)$.
- For the generation g, the reduction determines the shares of the parties as defined in $\mathbf{H}_1(\lambda, b)$, except that it replaces the keys $\kappa_i^{(g)} = y_i^{(g)}$ associated to all the non-input OR gates i such that $i \to h$ with one of the blocks from the challenge. In addition, we use the projective keys received from the challenger in order to generate all other pseudorandom blocks $y_i^{(g)}$ for $i \geq h$ that are needed by the sharing algorithm for this generation.
- For the last $\bar{g} - g$ generations, the reduction computes the shares of the players exactly as defined in $\mathbf{H}_1(\lambda, b)$.
- Finally, output whatever A outputs.

□

Lemma 19. $\{\mathbf{H}_2(\lambda, 0)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_2(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. The lemma follows by observing that in the experiment $\mathbf{H}_2(\lambda, b)$, for every generation $g \geq 2$, the ciphertext $\gamma_i^{(g)}$ associated to each of the unsatisfied output AND gates in $\hat{C}_g$ is uniform. By the validity of the adversary, all such gates are indeed unsatisfied, and thus the distribution of the shares given to the adversary is independent of the challenge bit. □

Theorem 9 follows by combining the above lemmas. □

We notice that the share of each *new* party within the generation $g \geq 1$ (i.e., any party that is not part of previous generations) consists of a master public key for a projective PRG outputting $m_g^\vee$ blocks of dimension λ, of a λ-bit seed for the standard PRG G, and of an encryption of the projective key[14] associated to each AND gate in the circuit $\hat{C}_g$. Recall that we increased the number of OR gates in the circuit $\hat{C}_g$ by at most n_g (in order to make sure that input wires only enter OR gates), and thus we can simply upper bound $m_g^\vee$ with m_g. Hence, by invoking Theorem 2, we obtain:

Corollary 4. *Let $\hat{\mathcal{A}}^{\mathsf{ckts}} = \{\hat{C}_g\}$ be the monotone circuits access structure, and denote by $m_g^\wedge$ (resp. $m_g^\vee$) the number of AND (resp. OR) gates in $\hat{C}_g$ for any $g \geq 1$. There exists a computationally private secret sharing scheme over $\mathcal{M} = \{0,1\}^\lambda$ for $\hat{\mathcal{A}}^{\mathsf{ckts}}$, from the following assumptions and with the following parameters:*

- *Under the RSA assumption, where the share size of all parties belonging to the generation $g \geq 1$ is $m_g^\wedge \cdot \mathsf{poly}(\lambda)$.*

[14] The key associated to the output AND gate actually equals the message, but this difference is immaterial when evaluating the share size.

- *Assuming iO and SSB hash functions, where the share size of all parties belonging to the generation $g \geq 1$ is $m_g^{\wedge} \cdot \mathsf{poly}(\lambda)$.*
- *Under either the DDH or the BDDH assumption, where the share size of all parties belonging to the generation $g \geq 1$ is $(m_g^{\vee})^2 \cdot \mathsf{poly}(\lambda) + m_g^{\wedge} \cdot O(\lambda)$.*
- *Under the LWE assumption, where the share size of all parties belonging to the generation $g \geq 1$ is $m_g^{\vee} \cdot \mathsf{poly}(\lambda) + m_g^{\wedge} \cdot O(\lambda)$.*

Remark 4 (On the number of gates)**.** *Recall that we assumed for simplicity that each input wire in the circuits $\hat{C}_g$ is only connected to OR gates with fan-in 1. For this to hold, one needs to add an AND gate with fan-in 1 for every input wire that goes into an OR gate with fan-in ≥ 2 (in order to maintain the invariant that the circuit alternates OR and AND layers). Hence, the number of AND gates $m_g^{\wedge}$ in the above corollary must be increased by an additive factor of at most n_g for every $g \geq 1$.*

5.4 CNF Formulas

Next, we consider the case of access structures expressed by monotone formulas in conjunctive normal form (CNF), i.e. $\varphi(x) = \bigwedge_{i=1}^{m} C_i$ where each clause C_i is a disjunction over a subset of the n variables $x = (x_1, \ldots, x_n)$.

We can easily adapt the above to the evolving setting. When the n-th party comes, the access structure $\mathcal{A}^{\mathsf{cnf}}$ evolves as follows: (i) The variable x_n corresponding to the n-th party can be added (using $\vee$) inside any of the already existing clauses $C_1, \ldots, C_m$; (ii) Additionally, we can remove any of the clauses $C_1, \ldots, C_m$. These conditions preserve monotonicity and cover all possible cases (indeed, monotonicity forbids to add new clauses to the CNF formula). Note that when only option (i) is available we get rigidity; in contrast, option (ii) violates rigidity. Consistently, we write $\hat{\mathcal{A}}^{m\text{-}\mathsf{cnf}}$ to denote the rigid evolving CNF formulas access structure (i.e., when only option (i) is available).

Construction 7

Let $\mathsf{pPRG} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.KeyGen}, \mathsf{pPRG.Eval})$ be a projective PRG with bounded polynomial stretch. Consider the following secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Setup}, \mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M} = \{0,1\}$ for the rigid evolving CNF formulas access structure $\hat{\mathcal{A}}^{m\text{-}\mathsf{cnf}}$, where m is the (fixed) number of clauses.

Sharing: At the onset, the dealer computes $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{pPRG.Setup}(1^{\lambda}, 1^{m})$. Let $(y_1, \ldots, y_m) = \mathsf{pPRG.Eval}(\mathsf{mpk}, \mathsf{msk})$. When the n-th party arrives, the dealer proceeds as follows:

- Let $\sigma_0 = \mu \oplus y_1 \oplus \cdots \oplus y_m$.
- Let $\mathcal{T}_n = \{j : x_n \in C_j\}$ and $\alpha_{\mathcal{T}_n} = \mathsf{pPRG.KeyGen}(\mathsf{mpk}, \mathsf{msk}, \mathcal{T}_n)$.
- Output $\sigma_n = (\mathsf{mpk}, \sigma_0, \alpha_{\mathcal{T}_n})$.

Reconstruction: The reconstruction algorithm $\mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{I}}, \mathcal{I})$, given the shares $(\sigma_i)_{i\in\mathcal{I}}$ such that $\sigma_i = (\mathsf{mpk}, \sigma_0, \alpha_{\mathcal{T}_i})$, and a minimal authorized subset $\mathcal{I} \in \hat{\mathcal{A}}_n^{m\text{-}\mathsf{cnf}}$, proceeds as follows for every $i \in [m]$:

- Let $j \in \mathcal{I}$ be a party that appears in the i-th clause.

- Compute y'_k, where y'_k is the k-th bit of $y' = \mathsf{pPRG.Eval}(\mathsf{mpk}, \alpha_{\mathcal{T}_j}, \mathcal{T}_j)$, such that the k-th index $j_k \in \mathcal{T}_i$ is equal to i.

Finally, output $\sigma_0 \oplus y'_1 \cdots \oplus y'_m$.

Let $\mathcal{I} \subseteq 2^{[n]}$ be a qualified subset $\mathcal{I} \in \hat{\mathcal{A}}_n^{m\text{-}\mathsf{cnf}}$ (recall that $\hat{\mathcal{A}}_n^{m\text{-}\mathsf{cnf}} \subseteq \hat{\mathcal{A}}^{m\text{-}\mathsf{cnf}}$ denotes the access structure with n parties), and let $x_{\mathcal{I}}$ be the corresponding satisfying assignment (i.e., $\varphi(x_{\mathcal{I}}) = 1$, where $x_{\mathcal{I}}$ is the assignment with value 1 in the positions indexed by $\mathcal{I}$, and 0 elsewhere). Correctness follows readily from the correctness property of the underlying projective PRG, since for every clause C_j in φ there exists a variable $x_i \in C_j$ such that $j \in \mathcal{T}_i$, and thus y_j can be computed by the key $\alpha_{\mathcal{T}_i}$ held by the parties in $\mathcal{I}$; thus, the parties in $\mathcal{I}$ can compute $(y'_1, \ldots, y'_m) = (y_1, \ldots, y_m)$. As for security, we have the following theorem.

Theorem 10. *Assuming* pPRG *is robust (Definition 14), the scheme* SS *described in Construction 7 is a computationally private secret sharing scheme over* $\{0,1\}$ *for the the rigid evolving CNF formulas access structure* $\hat{\mathcal{A}}^{m\text{-}\mathsf{cnf}}$*, for every fixed* $m \geq 1$*.*

Proof. Fix $m \geq 1$. Let n and $\mathcal{U} \notin \hat{\mathcal{A}}_n^{m\text{-}\mathsf{cnf}}$ be, respectively, the number of parties and the unqualified subset chosen by the adversary in the game defining computational privacy of SS. Also, let $\varphi(x) = \bigwedge_{i=1}^{m} C_i$ be the CNF formula made of m clauses and n variables which defines $\hat{\mathcal{A}}_n^{m\text{-}\mathsf{cnf}}$. Consider the following hybrid experiments:

$\mathbf{H}_0(\lambda, b)$**:** This experiment is identical to $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$.

$\mathbf{H}_1(\lambda, b)$ Identical to $\mathbf{H}_0(\lambda, b)$, except that we change how the challenger computes the shares of the parties. In particular, for every $i \in [m]$ such that $\mathcal{U} \cap C_i = \emptyset$, the challenger replaces y_i (i.e., one of the bits output by the projective PRG that are used to compute σ_0 which, in turn, is part of every share) with a random value $\rho \leftarrow\!\!\$\ \{0,1\}$.

Lemma 20. *For every* $b \in \{0,1\}$*, we have* $\{\mathbf{H}_0(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$*.*

Proof. First, note that $\mathcal{U} \cap C_i = \emptyset$ for some $i \in [m]$. Otherwise, the adversary is not valid since $\mathcal{U}$ would not be an unqualified set for $\hat{\mathcal{A}}_n^{m\text{-}\mathsf{cnf}}$. In turn, the condition $\mathcal{U} \cap C_i = \emptyset$ implies that $i \notin \mathcal{T}_j$ for every $j \in \mathcal{U}$ where $\mathcal{T}_j$ is the set defined in Construction 7. Thus, the adversary does not obtain any projective key which allows to compute y_i (i.e., the keys used to compute the encryption σ_0 of the message). Because of this, the robustness property of the projective PRG guarantees that y_i is indistinguishable from a random bit. This concludes the proof. □

Lemma 21. $\{\mathbf{H}_1(\lambda, 0)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$*.*

Proof. By the validity of the adversary, we have that $\varphi(x_{\mathcal{U}}) = 0$ where $x_{\mathcal{U}}$ is the assignment corresponding to the unqualified set of parties $\mathcal{U}$ chosen by the adversary. This implies that there exists $i \in [m]$ such that the i-th clause C_i of $\varphi(x)$ is not satisfied by $x_{\mathcal{U}}$, i.e., $\mathcal{U} \cap C_i = \emptyset$. Hence, the keys $(y_1, \ldots, y_m)$ (used to produce the ciphertext σ_0 encrypting the message) satisfies the following condition: $\exists i \in [m]$ such that $y_i = \rho \leftarrow\!\!\$\ \{0,1\}$. This implies that σ_0 (included in each share $(\sigma_1, \ldots, \sigma_n)$) is a one-time pad encryption of the message which, in turn, is identically distributed to a random string. Thus, the experiments $\mathbf{H}_1(\lambda, 0)$ and $\mathbf{H}_1(\lambda, 1)$ are identically distributed. □

Theorem 10 follows by combining the above lemmas. □

While Construction 7 only deals with message space $\mathcal{M} = \{0,1\}$, it is immediate to generalize it to $\mathcal{M} = \{0,1\}^\lambda$ by using a block projective PRG with bounded polynomial stretch. In turn, by invoking Theorem 2, we obtain:

Corollary 5. *For every fixed $m \geq 1$, there exists a computationally private secret sharing scheme over $\mathcal{M} = \{0,1\}^\lambda$ for the rigid evolving CNF formulas access structure $\hat{\mathcal{A}}^{m\text{-cnf}}$, from the following assumptions and with the following parameters:*

- *Under the RSA assumption, where the share size of party $n \geq 1$ is $\mathsf{poly}(\lambda)$.*
- *Assuming iO and SSB hash functions, where the share size of party $n \geq 1$ is $\mathsf{poly}(\lambda)$.*
- *Under either the DDH or the BDDH assumption, where the share size of party $n \geq 1$ is $m^2 \cdot \mathsf{poly}(\lambda)$.*
- *Under the LWE assumption, where the share size of party $n \geq 1$ is $m \cdot \mathsf{poly}(\lambda)$.*

Remark 5 (On removing rigidity)**.** *We observe that, assuming the underlying projective PRG has* unbounded *polynomial stretch, we can adapt Construction 7 to also work for the* non-rigid *evolving CNF formulas access structure $\mathcal{A}^{\mathsf{cnf}}$ (i.e., the number of clauses m now also grows). By Theorem 5, the latter requires to update the shares of the players. In this particular case, whenever the clause C_{m+1} is added to the CNF formula, it suffices to replace the value σ_0 with $\sigma_0 \oplus y_{m+1}$, where y_{m+1} is the next bit output by the projective PRG. Since the information σ_0 can be made public, this construction would only require* public *updates (without direct communication with the old players).*

From CNF formulas to truth tables. Following [1], given a truth table corresponding to a monotone function $f : \{0,1\}^n \to \{0,1\}$, we can associate to it a CNF formula with at most $m = 2^n$ clauses. The latter also applies to the evolving setting, taking into account monotonicity and rigidity. Thus, Construction 7 implies a secret sharing scheme for rigid evolving monotone truth tables, although the running time of the sharing algorithm is polynomial only for $n = O(\log \lambda)$. A negative result by Larsen and Simkin [27] shows that the latter limitation is inherent (even in the non-evolving setting) when the shares are succinct.

5.5 DNF Formulas

Consider the case of access structures expressed by monotone formulas in disjunctive normal form (DNF), i.e. $\varphi(x) = \bigvee_{i=1}^{m} C_i$ where each clause C_i is a conjunction over a subset of the n variables $x = (x_1, \ldots, x_n)$. In the evolving setting, we can neither remove old clauses nor add new variables to old clauses (as otherwise monotonocity is violated). In turn, we can always add clauses, and thus the value m is not fixed and grows with the number of parties n. However, rigidity requires that we never remove variables from old clauses, and that each new clause including old variables must also include at least one new variable.

To allow more flexibility, we will assume the clauses are not added one by one, but rather in generations of variable size. This distinction is important, as otherwise we might limit the way the access structure evolves. For instance, consider the case in which the first clause is $C_1(x_1, x_2) = x_1 \wedge x_2$ and then we add two additional clauses at the same time, say $C_2(x_1, x_2, x_3, x_4) = x_1 \wedge x_3 \wedge x_4$ and $C_3(x_1, x_2, x_3, x_4) = x_2 \wedge x_3 \wedge x_4$. In case the clauses can only be added one by one, rigidity forbids to add C_3 after C_2, as C_3 must necessarily contain at least one new variable x_5 which is needed for reconstruction.

We write $\hat{\mathcal{A}}^{\mathsf{dnf}}$ to represent the corresponding rigid evolving DNF formulas access structure. Also, we denote by m_g the number of clauses added to the DNF formula in generation $g \geq 1$,

and by n_g the number of variables added to the DNF formula in generation $g \geq 1$. This way, we have $n = \sum_g n_g$ and $m = \sum_g m_g$. We can obtain a secret sharing scheme for $\hat{\mathcal{A}}^{\mathsf{dnf}}$ by considering a special case of Construction 6 in which each circuit $\hat{C}_g$ consists of m_g output AND gates, which take as input a subset of the n inputs. Since there are no non-input OR gates in the circuits, we do not need a projective PRG anymore, and thus the share size of party $n \geq 1$ would be $\lambda \cdot (1 + m_g)$.

For completeness, we describe a slightly different construction (assuming only one-way functions) which optimizes the share size.

Construction 8

Let $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^*$ be a standard PRG with unbounded polynomial stretch. For a seed $\kappa \in \{0,1\}^\lambda$, we parse the output of $\mathsf{G}(\kappa)$ into blocks of size λ and denote with $\mathsf{G}(\kappa)[i]$ the i-th such block. Consider the following secret sharing scheme $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ over message space $\mathcal{M} = \{0,1\}^\lambda$ for the rigid evolving monotone DNF formulas access structure $\hat{\mathcal{A}}^{\mathsf{dnf}}$.

Sharing: When the generation $g \geq 1$ begins, the dealer proceeds as follows.

- Let $(C_j)_{j \in [m] \setminus [m-m_g]}$ be the new clauses, so that the current access structure is specified by the clauses $C_1, \ldots, C_m$.
- For every $i \in [n] \setminus [n-n_g]$ (i.e., the parties that have not received their shares yet), the dealer sets the share of party i to $\sigma_i = (\kappa_i, (\gamma_j)_{j \in \mathcal{T}_i})$, where $\kappa_i \leftarrow\!\!\$\ \{0,1\}^\lambda$, $\gamma_j = \mu \oplus \bigoplus_{x_i \in C_j} \mathsf{G}(\kappa_i)[j]$, and $\mathcal{T}_i$ is the set of the indexes $j \in [m] \setminus [m - m_g]$ such that $x_i \in C_j$ (i.e., the indexes of the clauses that contain the variable x_i associated to the i-th party).

Reconstruction: The reconstruction algorithm $\mathsf{SS.Recon}((\sigma_i)_{i \in \mathcal{I}}, \mathcal{I})$, given the shares $(\sigma_i)_{i \in \mathcal{I}}$ such that $\sigma_i = (\kappa_i, (\gamma_j)_{j \in \mathcal{T}_i})$, and a minimal authorized subset $\mathcal{I} \in \hat{\mathcal{A}}^{\mathsf{dnf}}_n$, proceeds as follows:

- Let $j^* \in [m]$ be the index corresponding to a clause C_{j^*} that is satisfied by $x_\mathcal{I}$, where $x_\mathcal{I}$ is the assignment associated to $\mathcal{I}$.
- For every $x_i \in C_{j^*}$, compute $y_i = \mathsf{G}(\kappa_i)[j^*]$.

Finally, output $\gamma_{j^*} \oplus \bigoplus_{i \in C_{j^*}} y_i$.

Correctness is immediate. As for security, we show the following result.

Theorem 11. *Assuming* G *is secure (Definition 1), the scheme* SS *described in Construction 8 is a computationally private secret sharing scheme over* $\mathcal{M} = \{0,1\}^\lambda$ *for the rigid evolving DNF formulas access structure* $\hat{\mathcal{A}}^{\mathsf{dnf}}$.

Proof. Let n and $\mathcal{U} \notin \hat{\mathcal{A}}^{\mathsf{dnf}}_n$ be, respectively, the number of parties and the unqualified subset chosen by the adversary in the game defining computational privacy of SS. Also, let $\varphi(x) = \bigvee_{i=1}^m C_i$ be the DNF formula made of m clauses and n variables which defines $\hat{\mathcal{A}}^{\mathsf{dnf}}_n$. Consider the following hybrid experiments:

$\mathbf{H}_0(\lambda, b)$**:** This experiment is identical to $\mathbf{G}^{\mathsf{priv}}_{\mathsf{SS},\mathsf{A}}(\lambda, b)$.

$\mathbf{H}_j(\lambda, b)$ **for** $j \in [n] \setminus \mathcal{U}$**:** Identical to $\mathbf{H}_{j-1}(\lambda, b)$, except that we change how the challenger computes the shares of the parties. In particular, for every $i \in [m]$ such that $x_j \in C_i$, and for every $k \in [n]$ such that $x_k \in C_i$, the challenger (during the computation of σ_k) lets $\gamma_i = \mu_b \oplus \bigoplus_{x_h \in C_i} y_h$ where $y_j \leftarrow\!\!\$\ \{0,1\}^\lambda$ (instead of $y_j = \mathsf{G}(\kappa_j)[i]$), and $y_h = \mathsf{G}(\kappa_h)[i]$ for every $x_h \in C_i$ such that $h \neq j$.

Lemma 22. *For every $j \in [n] \setminus \mathcal{U}$, and for every $b \in \{0,1\}$, it holds that $\{\mathbf{H}_{j-1}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_j(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. The only difference between $\mathbf{H}_{j-1}(\lambda, b)$ and $\mathbf{H}_j(\lambda, b)$ is in the computation of the ciphertexts $\{\gamma_i\}_{i\in[m]:x_j\in C_i}$ (note that γ_i is contained in the shares of party $k \in [n]$ only if $x_k \in C_i$). Since $j \notin \mathcal{U}$, we have that the seed κ_j of the j-th party is never revealed to the adversary. Hence, the lemma follows by the security of the PRG G (Definition 1). □

Lemma 23. $\{\mathbf{H}_{|[n]\setminus\mathcal{U}|}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_{|[n]\setminus\mathcal{U}|}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. By the validity of the adversary, we can conclude that $C_i \not\subseteq \mathcal{U}$ for every $i \in [m]$ (i.e., in each clause there is a variable associated to an honest party; otherwise, we would have $\varphi(x_\mathcal{U}) = 1$ where $x_\mathcal{U}$ is the assignment corresponding to the set $\mathcal{U}$). Thus, by definition of $\mathbf{H}_{|[n]\setminus\mathcal{U}|}(\lambda, 0)$ and $\mathbf{H}_{|[n]\setminus\mathcal{U}|}(\lambda, 1)$, for each $i \in [m]$, the ciphertext γ_i associated to the clause C_i is computed using at least one random key $y_j \leftarrow\!\!\$\ \{0,1\}^\lambda$ (for some $x_j \in C_i$ such that $j \in [n] \setminus \mathcal{U}$). Hence, all ciphertexts $\{\gamma_i\}_{i\in[m]}$ are uniformly distributed. This concludes the proof. □

Theorem 11 follows by combining the above lemmas. □

For every $n \geq 1$, the share size of the n-th party is $\lambda \cdot (1 + |\mathcal{T}_n|)$ where $|\mathcal{T}_n|$ is the number of clauses that contain the variable x_n associated to party n. Note that $|\mathcal{T}_n| \leq m_g$ (i.e., the number of new clauses added with the g-th generation). Hence, we obtain:

Corollary 6. *Assuming OWFs, there exists a computationally private secret sharing scheme over $\mathcal{M} = \{0,1\}^\lambda$ for the rigid evolving monotone DNF formulas access structure $\hat{\mathcal{A}}^{\mathsf{dnf}}$, where the share size of party $n \geq 1$ is $\lambda \cdot (1 + |\mathcal{T}_n|)$ and $|\mathcal{T}_n|$ is the number of clauses that contain the variable x_n associated to party n.*

6 Domain Extension

In the previous section, we gave constructions of computationally private secret sharing schemes for different evolving access structures, over domain $\mathcal{M} = \{0,1\}^\lambda$. In this section, we address the question of extending the domain to $\mathcal{M} = \{0,1\}^\ell$ for an arbitrary polynomial $\ell = \mathsf{poly}(\lambda)$.

Assume for simplicity that ℓ is a multiple of λ (if not, one can use padding). A trivial solution to the above question would be to consider ℓ parallel runs of an evolving secret sharing scheme for domain $\mathcal{M} = \{0,1\}^\lambda$. However, assuming the underlying secret sharing scheme has shares of size $s(\lambda, n, \lambda)$, the latter yields shares of size $\ell/\lambda \cdot s(\lambda, n, \lambda)$, and thus we are interested here in better solutions in terms of share size.

6.1 Evolving Information Dispersal

An essential tool for dealing with the question of domain extension for standard secret sharing is the concept of information dispersal, introduced by Krawczyk [26]. Intuitively, an information dispersal for the t-threshold access structure allows to distribute a message $\mu \in \{0,1\}^\ell$ to n parties, in such a way that any subset of t parties can reconstruct the message; the main

difference with threshold secret sharing is that unauthorized subsets of players could potentially learn partial information about the message.

The definitions below generalizes information dispersal to the setting of evolving access structures. While the treatment would make sense even for non-rigid evolving access structures, we stick to rigid ones as we do in the rest of the paper. An information dispersal $\mathsf{IDS} = (\mathsf{IDS.Share}, \mathsf{IDS.Recon})$ for evolving access structure $\mathcal{A}$, and with message space $\mathcal{M}$ and fragments space $\mathcal{C}$, consists of two polynomial-time algorithms specified as follows:

- The randomized sharing algorithm $\mathsf{IDS.Share}(\mu, (\gamma_i)_{i\in[n-1]}, \mathcal{A}_n)$ takes as input a message $\mu \in \mathcal{M}$, a collection of $n-1$ fragments $\gamma_1, \ldots, \gamma_{n-1} \in \mathcal{S}$, and an access structure $\mathcal{A}_n$ and outputs the fragment γ_n for the n-th party.
- The deterministic reconstruction algorithm $\mathsf{IDS.Recon}((\gamma_i)_{i\in\mathcal{I}}, \mathcal{I})$ takes as input a collection of fragments $(\gamma_i)_{i\in\mathcal{I}}$ along with a subset $\mathcal{I} \subseteq 2^{[m]}$ of the parties, and outputs a message $\mu' \in \mathcal{M}$.

Definition 21 (Correctness of evolving information dispersal)**.** *We say that an information dispersal* IDS *for an evolving access structure* $\mathcal{A}$ *is correct if for every message* $\mu \in \mathcal{M}$*, for every number of parties* $n \geq 1$*, and for every qualified subset* $\mathcal{I} \in \mathcal{A}_n$ *it holds that*

$$\mathbb{P}\left[\mu = \mu' : \begin{array}{c} \forall i \leq n : \gamma_i \leftarrow\!\!\$\ \mathsf{IDS.Share}(\mu, (\gamma_j)_{j<i}, \mathcal{A}_i) \\ \mu' = \mathsf{IDS.Recon}((\gamma_i)_{i\in\mathcal{I}}, \mathcal{I}) \end{array}\right] = 1.$$

An important goal in the design of an information dispersal is to minimize the maximum size of the fragments assigned to the parties (as otherwise, the information dispersal that sets $\gamma_i = \mu$ for all $i \in [n]$ always woks). Let $\mathcal{I} \in \mathcal{A}$ be an authorized subset, and let $|\mathcal{I}| = t$. Clearly, the size of each fragment assigned to the parties in $\mathcal{I}$ must be at least ℓ/t, and thus the maximum size of a fragment corresponds to authorized sets with minimal size. We call *optimal*, an information dispersal in which the maximum size of the fragments assigned to the players is equal to ℓ/t, where t is the minimum size of any authorized subset in $\mathcal{A}$. The theorem below says that whenever an evolving access structure is such that the minimal size of an authorized subset decreases over time, no information dispersal can be optimal in terms of the size of the fragments without updating the previous fragments.

Theorem 12. *Let* $\mathcal{A}$ *be an evolving access structure such that there exist indexes* n_1, n_2 *with* $n_1 < n_2$ *for which* $1 < t_2 < t_1$*, where* $t_1 = \min\{|\mathcal{I}| : \mathcal{I} \in \mathcal{A}_{n_1}\}$ *and* $t_2 = \min\{|\mathcal{I}| : \mathcal{I} \in \mathcal{A}_{n_2}\}$*. Then, any* optimal *information dispersal for* $\mathcal{A}$ *satisfying correctness requires to update the fragments of some of the parties.*

Proof. Let IDS be any information dispersal for the evolving access structure $\mathcal{A}$ in which the fragments of the parties never change once assigned by the dealer. Let t_1 be the minimal size of an authorized subset when party n_1 arrives. Since IDS is optimal, the maximum size of the fragments assigned to the parties when party n_1 arrives is ℓ/t_1. When party n_2 arrives, the minimal size of an authorized subset is $t_2 < t_1$ with $t_2 > 1$. Suppose that we assign to party n_2 a fragment of size ℓ/t_2; now, the parties belonging to any authorized subset $\mathcal{I} \in \mathcal{A}_{n_2}$ hold at most $\ell/t_2 + (t_2 - 1) \cdot \ell/t_1 < \ell/t_2 + (t_2 - 1) \cdot \ell/t_2 = \ell$ bits of information about the message, and thus cannot determine the message. □

Threshold access structures. In the non-evolving setting, an information dispersal for the t-threshold access structure is simply an erasure code, which can be instantiated using Reed-Solomon codes as follows. Parse the message μ into t blocks $\mu = (\mu_0, \ldots, \mu_{t-1})$, and interpret each block as an element of $\mathbb{GF}(q)$; if needed, the original message can be padded so that the

message length ℓ is a multiple of the threshold t. Hence, let $f(X) = \mu_0 + \mu_1 \cdot X + \cdots + \mu_{t-1} \cdot X^{t-1}$ be the polynomial over $\mathbb{GF}(q)$, whose coefficients are the fragments of the message. The fragment γ_i assigned to party $i \in [n]$ is $f(i)$. Now, any subset of t parties can successfully reconstruct the polynomial, and thus recover the message. Moreover, the size of each fragment is $\log q = \ell/t$, which is optimal.

We can extend the above to the evolving setting assuming the message is long enough, so that the size q can accommodate an arbitrary polynomial number of players $n = \mathsf{poly}(\lambda)$. In practice, this requires an exponentially large field and thus it is not very efficient as reconstruction takes quadratic (in t) time. Using more sophisticated encoding methods (e.g., LT codes [28]), one can support a large number of players for messages of arbitrary length more efficiently, with reconstruction time that is only linear (in t).

Theorem 13. *For every $n, \ell = \mathsf{poly}(\lambda)$ and $t = O(1)$, there exists an information dispersal over $\mathcal{M} = \{0,1\}^\ell$ for the evolving t-threshold access structure.*

Arbitrary access structures. As shown by Bèguin and Cresti [5], given an information dispersal for the threshold access structure, we can obtain an information dispersal for arbitrary access structures by setting the threshold t to the minimum size of an authorized subset. This optimizes the fragments size in terms of the maximum size of each fragment. The above works directly also in the evolving setting, by assuming an information dispersal for the evolving threshold access structure. We note that this construction requires that the dealer knows the minimum size of an authorized subset from the start; this property is satisfied by many evolving access structures such as graphs access structures, dynamic threshold access structures, and CNF formulas access structures.

Construction 9

Let $\mathsf{IDS}' = (\mathsf{IDS}'.\mathsf{Share}, \mathsf{IDS}'.\mathsf{Recon})$ be an information dispersal over message space $\mathcal{M}$ for the evolving threshold access structure, where the threshold is fixed. Consider the following information dispersal $\mathsf{IDS} = (\mathsf{IDS}.\mathsf{Share}, \mathsf{IDS}.\mathsf{Recon})$ over message space $\mathcal{M}$ for any evolving access structure $\mathcal{A}$ such that $t = \min\{|\mathcal{I}| : \mathcal{I} \in \mathcal{A}\}$.

Sharing: When the n-th party arrives, the dealer runs $\gamma_n = \mathsf{IDS}'.\mathsf{Share}(\mu, (\gamma_i)_{i<n}, t)$ and returns γ_n to the party.

Reconstruction: The reconstruction algorithm $\mathsf{IDS}.\mathsf{Recon}((\gamma_i)_{i\in\mathcal{I}}, \mathcal{I})$, given the fragments $(\gamma_i)_{i\in\mathcal{I}}$ corresponding to an authorized subset $\mathcal{I} \in \mathcal{A}_n$ reconstructs the message μ by running the reconstruction algorithm of the underlying information dispersal IDS'.

Theorem 14. *Assuming IDS' satisfies correctness, the information dispersal IDS described in Construction 9 satisfies correctness for any evolving access structure $\mathcal{A}$.*

Proof. The theorem simply follows by observing that any authorized subset $\mathcal{I} \in \mathcal{A}$ used for reconstruction satisfies $|\mathcal{I}| \geq t$. Hence, by correctness of IDS', running $\mathsf{IDS}'.\mathsf{Recon}((\gamma_i)_{i\in\mathcal{I}}, \mathcal{I})$ yields the message. □

While the information dispersal of Theorem 14 is optimal in terms of fragments size, it requires to know the minimum size of an authorized subset in advance. By Theorem 12, this limitation in inherent if one insists on obtaining an optimal information dispersal for arbitrary access structures.

6.2 Krawczyk's Compiler

Krawczyk [26] showed that, using an information dispersal for the threshold access structure, one can extend the domain of any secret sharing scheme for the same access structure in such a way that the maximum share size is $\ell/t + \mathsf{poly}(\lambda)$, where ℓ is the message length and t is the threshold. This is asymptotically optimal (as $\ell \to \infty$). Later, Bèguin and Cresti [5] showed how to extend this result to arbitrary access structures.

In this section, we show a simple generalization of the above constructions to the evolving setting. Namely, we assume an information dispersal for an arbitrary rigid evolving access structure $\hat{\mathcal{A}}$, and use it to extend the domain of any secret sharing scheme for the same access structure. Assuming the underlying information dispersal is optimal, the maximum share size of the transformed secret sharing scheme satisfies $s^*(\lambda, n, \ell) = s(\lambda, n, \lambda) + \ell/t$, where $s(\lambda, n, \lambda)$ is the share size of the underlying secret sharing scheme for messages of length λ, and t is the minimum size of an authorized set in $\hat{\mathcal{A}}$.

Construction 10

Let $\mathsf{SS} = (\mathsf{SS.Share}, \mathsf{SS.Recon})$ be a secret sharing scheme over message space $\{0,1\}^\lambda$ for the rigid evolving access structure $\hat{\mathcal{A}}$, let $\mathsf{IDS} = (\mathsf{IDS.Share}, \mathsf{IDS.Recon})$ be an information dispersal over message space $\{0,1\}^\ell$ for $\hat{\mathcal{A}}$, and let $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a secret-key encryption scheme with message and ciphertext space $\{0,1\}^\ell$. Consider the following secret sharing scheme $\mathsf{SS}^* = (\mathsf{SS}^*.\mathsf{Share}, \mathsf{SS}^*.\mathsf{Recon})$ over message space $\{0,1\}^\ell$ for $\hat{\mathcal{A}}$.

Sharing: At the onset, the dealer samples $\kappa \leftarrow\!\!\$\ \{0,1\}^\lambda$ and lets $\gamma = \mathsf{SKE.Enc}(\kappa, \mu)$. When the n-th party arrives, the dealer runs $\sigma_n = \mathsf{SS.Share}(\kappa, (\sigma_i)_{i<n}, \hat{\mathcal{A}}_n)$ and $\gamma_n = \mathsf{IDS.Share}(\gamma, (\gamma_i)_{i<n}, \hat{\mathcal{A}}_n)$. Finally, it outputs $\sigma_n^* = (\sigma_n, \gamma_n)$.

Reconstruction: The reconstruction algorithm $\mathsf{SS}^*.\mathsf{Recon}((\sigma_i^*)_{i\in\mathcal{I}}, \mathcal{I})$, given the shares $(\sigma_i^*)_{i\in\mathcal{I}}$ such that $\sigma_i^* = (\sigma_i, \gamma_i)$, and an authorized subset $\mathcal{I} \in \hat{\mathcal{A}}_n$, it returns $\mathsf{Dec}(\kappa, \gamma)$ where $\kappa = \mathsf{SS.Recon}((\sigma_i)_{i\in\mathcal{I}}, \mathcal{I})$ and $\gamma = \mathsf{IDS.Recon}((\gamma_i)_{i\in\mathcal{I}}, \mathcal{I})$.

Correctness follows immediately by the correctness of the underlying building blocks. As for security, we establish the following result.

Theorem 15. *Assuming that* SS *is a computationally private secret sharing scheme for the rigid evolving access structure* $\hat{\mathcal{A}}$ *(Definition 19), and that* SKE *is one-time secure (Definition 5), the scheme* SS^* *described in Construction 10 is a computationally private secret sharing scheme for* $\hat{\mathcal{A}}$.

Proof. Let $\mathbf{H}_0(\lambda, b) \equiv \mathbf{G}^{\mathsf{priv}}_{\mathsf{SS}^*,\mathsf{A}^*}(\lambda, b)$ be the original experiment defining computational privacy of the secret sharing scheme SS^*, with the challenge bit fixed to $b \in \{0,1\}$. We consider the following hybrid experiment.

$\mathbf{H}_1(\lambda, b)$: This experiment is identical to $\mathbf{H}_0(\lambda, b)$, except that the challenger samples two independent keys $\kappa, \kappa' \leftarrow\!\!\$\ \{0,1\}^\lambda$ and the shares σ_n are now computed using κ', whereas the ciphertext γ (and thus the fragments γ_n) is still computed using κ as in the original experiment.

Lemma 24. *For all* $b \in \{0,1\}$*, it holds that* $\mathbf{H}_0(\lambda, b) \approx_c \mathbf{H}_1(\lambda, b)$.

Proof. Fix $b \in \{0,1\}$. The proof is by reduction to the computational privacy of the underlying secret sharing scheme SS. By contradiction, assume that there exists a PPT adversary A^* that can distinguish between $\mathbf{H}_0(\lambda, b)$ and $\mathbf{H}_1(\lambda, b)$ with non-negligible probability. We construct a PPT adversary A that breaks computational privacy of SS as follows:

- Run A^* and receive back a pair of messages (μ_0, μ_1), an integer $n \in \mathbb{N}$, and an unauthorized subset $\mathcal{U} \notin \hat{\mathcal{A}}_n$.
- Sample $\kappa, \kappa' \leftarrow\!\!\$\ \{0,1\}^\lambda$, and forward (κ, κ'), n and $\mathcal{U}$ to the challenger.
- Upon receiving the challenge shares $(\sigma_i)_{i\in\mathcal{U}}$, compute $\gamma = \mathsf{SKE.Enc}(\kappa, \mu_b)$ and for each $i \in [n]$ generate $\gamma_i = \mathsf{IDS.Share}(\gamma, (\gamma_j)_{j<i}, \hat{\mathcal{A}}_i)$. Return $(\sigma_i^*)_{i\in\mathcal{U}} = ((\sigma_i)_{i\in\mathcal{U}}, (\gamma_i)_{i\in\mathcal{U}})$ to A^*.
- Upon receiving a bit b' from A^* return b' to the challenger.

For the analysis, we note that when the shares $(\sigma_i)_{i\in\mathcal{U}}$ are generated by a secret sharing of κ, the distribution generated by the reduction is identical to the view of A^* in a run of experiment $\mathbf{H}_0(\lambda, b)$. Similarly, when the shares $(\sigma_i)_{i\in\mathcal{U}}$ are generated by a secret sharing of κ', the distribution generated by the reduction is identical to the view of A^* in a run of experiment $\mathbf{H}_1(\lambda, b)$. The lemma follows. □

Lemma 25. $\mathbf{H}_1(\lambda, 0) \approx_c \mathbf{H}_1(\lambda, 1)$.

Proof. The proof is by reduction to the one-time security of the underlying secret-key encryption scheme SKE. By contradiction, assume that there exists a PPT adversary A^* that can distinguish between $\mathbf{H}_1(\lambda, 0)$ and $\mathbf{H}_1(\lambda, 1)$ with non-negligible probability ϵ. We construct a PPT adversary A that breaks one-time security of SKE as follows:

- Run A^* and receive back a pair of messages (μ_0, μ_1), an integer $n \in \mathbb{N}$, and an unauthorized subset $\mathcal{U} \notin \hat{\mathcal{A}}_n$.
- Forward (μ_0, μ_1) to the challenger, and receive back the ciphertext γ.
- Sample $\kappa' \leftarrow\!\!\$\ \{0,1\}^\lambda$, and for each $i \in [n]$ generate $\sigma_i = \mathsf{SS.Share}(\kappa', (\sigma_j)_{j<i}, \hat{\mathcal{A}}_i)$, as well as $\gamma_i = \mathsf{IDS.Share}(\gamma, (\sigma_j)_{j<i}, \hat{\mathcal{A}}_i)$. Return $(\sigma_i^*)_{i\in\mathcal{U}} = ((\sigma_i)_{i\in\mathcal{U}}, (\gamma_i)_{i\in\mathcal{U}})$ to A^*.
- Upon receiving a bit b' from A^* return b' to the challenger.

For the analysis, we note that when the challenge ciphertext γ is generated by encrypting μ_0, the distribution generated by the reduction is identical to the view of A^* in a run of experiment $\mathbf{H}_1(\lambda, 0)$. Similarly, when the challenge ciphertext γ is generated by encrypting μ_1, the distribution generated by the reduction is identical to the view of A^* in a run of experiment $\mathbf{H}_1(\lambda, 1)$. The lemma follows. □

The theorem statement now follows by combining the above lemmas. □

7 Conclusions

We have initiated a systematic study of *evolving* secret sharing schemes in the *computational* setting. Our main finding is that switching to computational security allows to obtain secret sharing schemes for a plethora of evolving access structures, including dynamic threshold, graphs, CNF and DNF formulas, and monotone circuits access structures. Furthermore, for many of there access structures, our secret sharing schemes are *succinct*, i.e., much smaller compared to the size of a natural computational representation of the evolving access structure.

A first natural direction for future research would be to obtain secret sharing schemes for more evolving access structures (e.g., monotone NP [24]), or to improve our constructions in terms of hardness assumptions and/or share size. Another interesting open problem is to study evolving secret sharing in the context of *adaptive security* [22], or with additional properties such as verifiability [3], and non-malleability [19, 17].

Acknowledgements. The first author was supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the second author was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU and by Sapienza University under the project SPECTRA.

References

[1] Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. Cryptology ePrint Archive, Paper 2023/955 (2023), https://eprint.iacr.org/2023/955

[2] Applebaum, B., Nir, O.: Upslices, downslices, and secret-sharing with complexity of 1.5^n. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 627–655. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9_21

[3] Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 590–609. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_32

[4] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im) possibility of obfuscating programs. Journal of the ACM (JACM) **59**(2), 1–48 (2012)

[5] Béguin, P., Cresti, A.: General short computational secret sharing schemes. In: Guillou, L.C., Quisquater, J.J. (eds.) EUROCRYPT'95. LNCS, vol. 921, pp. 194–208. Springer, Heidelberg (May 1995). https://doi.org/10.1007/3-540-49264-X_16

[6] Beimel, A.: Secret-sharing schemes: A survey. In: Coding and Cryptology - Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings. vol. 6639, pp. 11–46. Springer (2011)

[7] Beimel, A., Farràs, O., Mintz, Y.: Secret-sharing schemes for very dense graphs. Journal of Cryptology **29**(2), 336–362 (Apr 2016). https://doi.org/10.1007/s00145-014-9195-8

[8] Beimel, A., Othman, H.: Evolving ramp secret-sharing schemes. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 313–332. Springer, Heidelberg (Sep 2018). https://doi.org/10.1007/978-3-319-98113-0_17

[9] Beimel, A., Othman, H.: Evolving ramp secret sharing with a small gap. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 529–555. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45721-1_19

[10] Blakley, G.R.: Safeguarding cryptographic keys. Proceedings of AFIPS 1979 National Computer Conference **48**, 313–317 (1979)

[11] Cachin, C.: On-line secret sharing. In: Boyd, C. (ed.) 5th IMA International Conference on Cryptography and Coding. LNCS, vol. 1025, pp. 190–198. Springer, Heidelberg (Dec 1995)

[12] Csirmaz, L.: The size of a share must be large. In: Santis, A.D. (ed.) EUROCRYPT'94. LNCS, vol. 950, pp. 13–22. Springer, Heidelberg (May 1995). https://doi.org/10.1007/BFb0053420

[13] Csirmaz, L.: The size of a share must be large. Journal of Cryptology **10**(4), 223–231 (Sep 1997). https://doi.org/10.1007/s001459900029

[14] Csirmaz, L., Tardos, G.: On-line secret sharing. Des. Codes Cryptogr. **63**(1), 127–147 (2012)

[15] Desmedt, Y., Dutta, S., Morozov, K.: Evolving perfect hash families: A combinatorial viewpoint of evolving secret sharing. In: Mu, Y., Deng, R.H., Huang, X. (eds.) CANS 19. LNCS, vol. 11829, pp. 291–307. Springer, Heidelberg (Oct 2019). https://doi.org/10.1007/978-3-030-31578-8_16

[16] Dutta, S., Roy, P.S., Fukushima, K., Kiyomoto, S., Sakurai, K.: Secret sharing on evolving multi-level access structure. In: You, I. (ed.) WISA 19. LNCS, vol. 11897, pp. 180–191. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-39303-8_14

[17] Faonio, A., Venturi, D.: Non-malleable secret sharing in the computational setting: Adaptive tampering, noisy-leakage resilience, and improved rate. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 448–479. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26951-7_16

[18] Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: 21st ACM STOC. pp. 25–32. ACM Press (May 1989). https://doi.org/10.1145/73007.73010

[19] Goyal, V., Kumar, A.: Non-malleable secret sharing. In: Diakonikolas, I., Kempe, D., Henzinger, M. (eds.) 50th ACM STOC. pp. 685–698. ACM Press (Jun 2018). https://doi.org/10.1145/3188745.3188872

[20] Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Roughgarden, T. (ed.) ITCS 2015. pp. 163–172. ACM (Jan 2015). https://doi.org/10.1145/2688073.2688105

[21] Ito, M., Saito, A., Nishizeki, T.: Secret sharing schemes realizing general access structure. In: Proc. IEEE Global Telecommunication Conf. (Globecom'87). pp. 99–102 (1987)

[22] Jafargholi, Z., Kamath, C., Klein, K., Komargodski, I., Pietrzak, K., Wichs, D.: Be adaptive, avoid overcommitting. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 133–163. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7_5

[23] Komargodski, I., Naor, M., Yogev, E.: How to share a secret, infinitely. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 485–514. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_19

[24] Komargodski, I., Naor, M., Yogev, E.: Secret-sharing for NP. Journal of Cryptology **30**(2), 444–469 (Apr 2017). https://doi.org/10.1007/s00145-015-9226-0

[25] Komargodski, I., Paskin-Cherniavsky, A.: Evolving secret sharing: Dynamic thresholds and robustness. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. LNCS, vol. 10678, pp. 379–393. Springer, Heidelberg (Nov 2017). https://doi.org/10.1007/978-3-319-70503-3_12

[26] Krawczyk, H.: Secret sharing made short. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 136–146. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_12

[27] Larsen, K.G., Simkin, M.: Secret sharing lower bound: Either reconstruction is hard or shares are long. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 566–578. Springer, Heidelberg (Sep 2020). https://doi.org/10.1007/978-3-030-57990-6_28

[28] Luby, M.: Lt codes. In: 43rd FOCS. pp. 271–282. IEEE Computer Society Press (Nov 2002). https://doi.org/10.1109/SFCS.2002.1181950

[29] Luby, M., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A.: Efficient erasure correcting codes. IEEE Trans. Inf. Theory **47**(2), 569–584 (2001)

[30] Mazor, N.: A lower bound on the share size in evolving secret sharing. In: 4th Conference on Information-Theoretic Cryptography, ITC 2023, June 6-8, 2023, Aarhus University, Aarhus, Denmark. LIPIcs, vol. 267, pp. 2:1–2:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)

[31] Mitzenmacher, M.: Digital fountains: a survey and look forward. In: 2004 IEEE Information Theory Workshop, San Antonio, TX, USA, 24-29 October, 2004. pp. 271–276. IEEE (2004)

[32] Okamoto, T., Pietrzak, K., Waters, B., Wichs, D.: New realizations of somewhere statistically binding hashing and positional accumulators. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 121–145. Springer, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48797-6_6

[33] Paskin-Cherniavsky, A.: How to infinitely share a secret more efficiently. Cryptology ePrint Archive, Report 2016/1088 (2016), https://eprint.iacr.org/2016/1088

[34] Pueyo, I.C., Cramer, R., Xing, C.: Bounds on the threshold gap in secret sharing and its applications. IEEE Trans. Inf. Theory **59**(9), 5600–5612 (2013)

[35] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC. pp. 73–85. ACM Press (May 1989). https://doi.org/10.1145/73007.73014

[36] Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the Association for Computing Machinery **21**(2), 120–126 (Feb 1978). https://doi.org/10.1145/359340.359342

[37] Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979)

[38] Shokrollahi, M.A., Luby, M.: Raptor codes. Found. Trends Commun. Inf. Theory **6**(3-4), 213–322 (2009)

[39] Vinod, V., Narayanan, A., Srinathan, K., Rangan, C.P., Kim, K.: On the power of computational secret sharing. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 162–176. Springer, Heidelberg (Dec 2003)

[40] Xing, C., Yuan, C.: Evolving secret sharing schemes based on polynomial evaluations and algebraic geometry codes. Cryptology ePrint Archive, Report 2021/1115 (2021), https://eprint.iacr.org/2021/1115