Electronic Colloquium on Computational Complexity, Report No. 91 (2023)

Succinct Computational Secret Sharing[*]

Benny Applebaum, Tel Aviv University
Amos Beimel, Ben-Gurion University
Yuval Ishai, Technion
Eyal Kushilevitz, Technion
Tianren Liu, Peking University
Vinod Vaikuntanathan, MIT

June 18, 2023

Abstract

A secret-sharing scheme enables a dealer to share a secret s among n parties such that only authorized subsets of parties, specified by a monotone *access structure* $f : \{0,1\}^n \to \{0,1\}$, can reconstruct s from their shares. Other subsets of parties learn nothing about s.

The question of minimizing the (largest) *share size* for a given f has been the subject of a large body of work. However, in most existing constructions for general access structures f, the share size is not much smaller than the size of some natural computational representation of f, a fact that has often been referred to as the "representation size barrier" in secret sharing.

In this work, we initiate a systematic study of *succinct computational secret sharing* (SCSS), where the secrecy requirement is computational and the goal is to substantially beat the representation size barrier. We obtain the following main results.

- SCSS via Projective PRGs. We introduce the notion of a *projective PRG*, a pseudorandom generator for which any subset of the output bits can be revealed while keeping the other output bits hidden, using a *short* projective seed. We construct projective PRGs with different levels of succinctness under a variety of computational assumptions, and apply them towards constructing SCSS for graph access structures, monotone CNF formulas, and (less succinctly) useful subclasses of monotone circuits and branching programs. Most notably, under the sub-exponential RSA assumption, we obtain a SCSS scheme that, given an *arbitrary* access structure f, represented by a truth table of size $N = 2^n$, produces shares of size $\mathrm{polylog}(N) = \mathrm{poly}(n)$ in time $\tilde{O}(N)$. For comparison, the share size of the best known information-theoretic schemes is $O(N^{0.58})$.
- SCSS via One-way Functions. Under the (minimal) assumption that one-way functions exist, we obtain a near-quadratic separation between the total share size of computational and information-theoretic secret sharing. This is the strongest separation one can hope for, given the state of the art in secret sharing lower bounds. We also construct SCSS schemes from one-way functions for useful classes of access structures, including forbidden graphs and monotone DNF formulas. This leads to constructions of fully-decomposable conditional disclosure of secrets (also known as privacy-free garbled circuits) for general functions, represented by a truth table of size $N = 2^n$, with share size $\mathrm{polylog}(N)$ and computation time $\tilde{O}(N)$, assuming sub-exponentially secure one-way functions.

[*]This is the full version of a paper that appears in STOC'23.

ISSN 1433-8092

Contents

1 Introduction

Secret sharing [Sha79, Bla79, ISN87] is a powerful cryptographic technique that allows a dealer to distribute shares of a secret to n parties such that certain authorized subsets of parties can recover the secret, while unauthorized subsets do not learn any information about it. More formally, a secret-sharing scheme realizing a *monotone access structure* (a monotone function) $f : \{0,1\}^n \to \{0,1\}$ is a randomized algorithm that, given a secret s, outputs n shares $\mathsf{sh}_1, \dots, \mathsf{sh}_n$ such that the following properties hold:

- CORRECTNESS: for any $(x_1, \dots, x_n) \in \{0,1\}^n$ where $f(x_1, \dots, x_n) = 1$, corresponding to an "authorized subset" of parties $T = \{P_i : x_i = 1\}$, the secret s can be recovered given the (authorized) collection of shares $(\mathsf{sh}_i)_{i:x_i=1}$;
- (INFORMATION-THEORETIC) SECRECY: for any $(x_1, \dots, x_n) \in \{0,1\}^n$ where $f(x_1, \dots, x_n) = 0$, the (unauthorized) collection of shares $(\mathsf{sh}_i)_{i:x_i=1}$ reveal nothing about the secret.

Throughout this work, we will assume that the secret s is a single bit, and focus on the *share size*, defined as the bit-length of the largest share sh_i, as our main complexity measure. (It will sometimes be convenient to consider the *total* share size instead.) The question of minimizing the share size for a given f has been the subject of a large body of work, spanning more than three decades. However, in most existing constructions for general access structures f, the share size is not much smaller than the size of some natural computational representation of the access structure f, be it monotone CNF or DNF formulas, monotone circuits or branching programs, or a graph. This phenomenon has often been referred to as the "representation size barrier" in secret sharing.

In a prototypical scenario where the access structure f is an arbitrary monotone function, natural constructions from the 1990s achieved a share size of 2^n bits, and even slightly better, namely $O(2^n/\sqrt{n})$ bits [BL90]. This number corresponds to the size of the smallest possible monotone formulas for worst-case functions f. A construction based on monotone span programs was presented in [KW93]. This construction yields better secret-sharing schemes for some functions [BGW99]; however it does not improve the share size for arbitrary functions. We have also known from the 90s that there is an (explicit) monotone function f that requires a *total* share size of $\Omega(n^2/\log n)$ bits [Csi97, Csi96], a far cry from the upper bound. No better lower bounds are known for general secret-sharing schemes, even in a non-explicit sense.[1]

The first improvement to this state of affairs came with the works of Liu, Vaikuntanathan, and Wee [LVW18, LV18], that achieved a share size of $2^{0.994n}$. This has since been improved to $(1.5)^{n+o(n)} < 2^{0.585n}$ in a recent sequence of exciting results [ABF+19, ABNP20, AN21]. Notwithstanding this excitement, Applebaum, Beimel, Nir, Peter, and Pitassi [ABN+22] have recently shown that the approaches in these works *cannot* achieve a sub-exponential share size, in fact even shares of size $2^{o(n/\log^2 n)}$. This is the starting point of our work.

Breaking the Impasse with Computational Assumptions. Modern cryptography was born out of the ashes of Shannon's seminal work from the 1940s, which showed that encrypting n bits

[1] Better lower bounds *are* known for the very special class of linear secret-sharing schemes, where the reconstruction algorithm, for every subset of parties, is restricted to be a linear function of the shares, see e.g., [BGW99, PR17].

required keys of size at least n bits, by moving to the world of computational security where secrecy is guaranteed only against time-bounded adversaries. In the hopes of repeating this success story, it is natural to ask:

Can computational cryptography help break the impasse in secret sharing?

Secret-sharing schemes with computational security have been considered previously (see Section 1.2); however, the previous computational constructions fail to achieve this goal. Specifically, the following concrete questions beg for our attention:

1. Super-polynomial improvements for high-end functions? Consider an *arbitrary* monotone function f over n bits whose description length is close to the truth-table size $N = 2^n$. As already mentioned, information-theoretic constructions achieve polynomial savings and yield secret sharing schemes with shares of size N^c for a constant $c < 1$. Is it possible to obtain super-polynomial savings in N by moving to computational security? Ideally, we would like to reduce the share size to be polylogarithmic in the truth-table size, i.e., $\mathrm{poly}(n) = \mathrm{poly}(\log N)$.

2. Polynomial improvements for efficient functions? The recent line of information-theoretic constructions does not seem to yield significantly-better secret-sharing schemes for "efficiently computable" functions. For example, even if a function f admits a polynomial-size formula, say of size n^3, we still do not know how to realize it with total share size of, say, $n^{2.99}$. The situation is essentially similar with respect to computational secret-sharing schemes. While computational assumptions help us handle functions f that can be represented by stronger, more expressive, computational models such as monotone circuits [Yao89] or even monotone NP-relations [KNY17], the share size in these schemes grows at least linearly with the description of f. The question of realizing functions that are $S(n)$-time computable with respect to some natural expressive computational model (say monotone circuits) with share size polynomially smaller than S, has so far remained open.

3. Separating computationally secure schemes from information-theoretic schemes? It is known that computational secret sharing can lead to savings when the secret is long [Kra94].[2] However, in the case of single-bit secrets, where the cost is due to the "structure" of the function f, we do not have such a separation, and we do not know whether computational assumptions buy us anything beyond the power of information-theoretic tools.

To capture these questions, we initiate a systematic study of *succinct computational secret-sharing schemes* (SCSS for short, pronounced "success") that, given a description of a function f, under some representation model (e.g., truth tables, graphs, CNFs, DNFs, or monotone circuits), realizes f with share size smaller than the representation of f. We construct succinct computational secret-sharing schemes for various classes of access structures, breaking the information-theoretic impasse, and

[2]In fact, the existence of encryption schemes for long messages with short keys that bypass Shannon's lower-bound also (weakly) demonstrates such a separation. To see this, observe that any symmetric-encryption scheme gives rise to a "2-out-of-2" secret-sharing scheme in which one party receives the key and the other party receives the ciphertext. (Formally, the underlying function f is an AND of 2 bits.) Shannon's lower bound says that, information-theoretically, each share must be as long as the message, whereas computationally-secure encryption beats this bound.

providing affirmative answers to all the above questions. For example, one of our signature results is the following:

Theorem 1.1 (SCSS for truth tables, Informal). *Under the sub-exponential RSA assumption, for every monotone function $f : \{0,1\}^n \to \{0,1\}$, represented as a truth table of size $N = 2^n$, there exists a computational secret-sharing scheme with share size $\mathrm{polylog}(N) = \mathrm{poly}(n)$. The scheme is secure against any $\mathrm{poly}(N)$-time adversary, and its sharing and reconstruction algorithms run in time $\tilde{O}(N)$.*

Alternatively, under the standard RSA assumption, we can achieve share size N^ε for any constant $\varepsilon > 0$. Note that here, and in the rest of the introduction, we simplify the security definition by considering the representation size N to also be a security parameter. If we separate between the two parameters (as in Definition 2.5), the shares have size $\mathrm{poly}(\lambda, \log N)$. See Remark 2.6 for further discussion.

Several of our constructions are enabled by a new cryptographic primitive called a *projective pseudorandom generator* (projective PRG, or pPRG for short), which we define and construct from several standard "public-key" type assumptions including the RSA assumption. We describe these, and additional results that assume only one-way functions, in the sequel.

1.1 Our Results and Techniques

We now proceed to describe our results in greater detail.

1.1.1 pPRG-based Constructions

Projective pseudorandom generators. Our main tool for constructing SCSS is a new cryptographic primitive called a *projective pseudorandom generator* (abbreviated pPRG) that we define and construct. A pPRG expands a short seed into a longer pseudorandom string for which any subset of the bits of the pseudorandom string can be revealed without disclosing any information about the other bits of the string. Of course, this can be accomplished by simply giving the subset of the output bits; however, we require that this is done using a *short* projective key (or seed). More formally, a pPRG consists of 3 algorithms: (1) a setup algorithm, which outputs a master key that can be used to generate a longer pseudorandom string, (2) a key generation algorithm that, given the master key and a set T, outputs a *short* projective key that is associated with this set, and (3) an evaluation algorithm that, given T and the projective key associated with T, outputs the subset T of the output bits of the pseudorandom string. We require a pPRG to be *robust* in the sense that an adversary who chooses sets $T_1, \ldots, T_\ell$ and is given their respective projective keys cannot learn information on the output bits not in $T_1 \cup \cdots \cup T_\ell$. (See Section 3 for the formal definition.) We construct a pPRG with short projective keys under the sub-exponential RSA assumption.

Theorem 1.2 (PPRG from RSA, informal). *Under the sub-exponential (resp., polynomial) RSA assumption, there exists a sub-exponentially-robust pPRG (resp., polynomially-robust pPRG) with sub-exponential stretch (resp., arbitrary polynomial stretch) whose projective keys and public parameters are both fully succinct, i.e., of length $\mathrm{poly}(k, \log m)$, where m is the output length and k the input length.*

Our pPRG from the RSA assumption is based on a construction of [AIKW15] of an online-efficient randomized encoding based on the RSA assumption. However, the original construction suffers from long public parameters. Specifically, for a pPRG with output length m, the public parameters contain m random primes. In order to prove Theorem 1.2, we have to reduce the length of the public parameters to $\mathrm{polylog}(m)$.

The idea is to generate the primes in a "pseudorandom" way, using a sampler algorithm that maps a short random tape to the m primes; we provide the short tape as part of the public parameters. This generation should be done with some care since the pPRG adversary gets to see the *random tape* as part of the public parameters. It turns out that the security reduction to RSA goes through as long as the sampler satisfies the following properties: (1) except with negligible probability, the outputs of the sampler are m distinct primes; (2) the marginal distribution of every single prime is uniform over the set of primes whose length is the security parameter; and (3) there exists an efficient *planting* procedure that, given a prime e and an index i, samples a random tape that generates e as its i^{th} output. We present a simple construction of such a sampler with a random tape of size $\mathrm{polylog}(m)$ using a k-wise independent distribution for an appropriate k. As a result, the total length of the public parameters is small, and can be even integrated as part of the projective keys. See Section 3.2 for a full description of this construction.

Perhaps unsurprisingly, strong cryptographic primitives such as indistinguishability obfuscation (iO) give easier routes towards SCSS. For instance, general-purpose pseudorandom correlation generators based on iO can be used to generically compress the shares of existing (non-succinct) secret-sharing schemes. This requires a long but reusable common random string [HIJ+16, BCG+19, ASY22] (building on [HW15]), which can be compressed using a random oracle. In Section 3.3, we describe a simple construction of a pPRG from iO and somewhere statistically binding (SSB) hash functions [HW15]. Combined with the iO construction of [JLS22] and constructions of SSB hash functions [OPWW15], this implies a succinct pPRG based on (the conjunction of) three standard assumptions that do not include RSA.

Theorem 1.3 (PPRG from IO and SSB hash functions, informal). *Under the assumption that indistinguishability obfuscation schemes and somewhere statistically binding hash functions with sub-exponential (resp., polynomial) security exist, there exists a sub-exponentially-robust pPRG (resp., polynomially-robust pPRG) with sub-exponential stretch (resp., arbitrary polynomial stretch) whose projective keys and public parameters are both fully succinct, i.e., of length* $\mathrm{poly}(k, \log m)$*, where* m *is the output length and* k *the input length.*

Finally, we also construct pPRGs with weaker parameters based on other assumptions: the decisional Diffie-Hellman assumption, with projective keys of size $O(\lambda)$ where λ is the security parameter, and public parameters of size $O(m^2)$); the decisional bilinear Diffie-Hellman assumption, with projective keys of size $O(m^\delta)$ and public parameters of size $O(m^{2(1-\delta)})$ for every constant δ, with the additional property that the public parameters are independent of the pPRG seed and are reusable; and the learning with errors assumption, with projective keys of size $O(\lambda)$ and public parameters of size $O(m)$. Since the public parameters in these constructions are not succinct, they are insufficient to derive Theorem 1.2, though they do yield some non-trivial SCSS results for special representations such as graphs. This is an uncommon situation in cryptography, namely, a natural task for which we have an efficient construction from the RSA assumption but seemingly not from

Representation	Information-theoretic secret sharing	Computational secret sharing	Computational assumption
Truth tables of size $N = 2^n$	$N^{0.585} = 2^{0.585n}$ [AN21]	$\text{polylog}(N) = \text{poly}(n)$	pPRG
CNF formulas with m clauses	$O(m)$ [ISN87]	$\text{polylog}(m)$	pPRG
Graphs with n vertices	$O(n/\log n)$ [EP97]	$\text{polylog}(n)$	pPRG
Partite functions with truth table size N	$2^{\tilde{O}(\sqrt{n})}$ [LVW18]	$\text{polylog}(N) = \text{poly}(n)$	OWF
Forbidden graphs with n vertices	$2^{\tilde{O}(\sqrt{\log n})}$ [LVW17]	$\text{polylog}(n)$	OWF
Csirmaz's access structure (total share size)	$\Theta(n^2/\log n)$ [Csi96]	$\tilde{O}(n)$	OWF

Table 1: Summary of our main SCSS constructions. We compare the share size to the best known information-theoretic secret-sharing schemes. For each construction we list the assumption. We assume sub-exponential security by default. We will let $N = 2^n$. Recall that pPRGs can be based either on RSA or on iO and SSB hash functions.

other concrete assumptions, even ones as powerful as bilinear maps and LWE. We defer further description of these constructions to the full version of this paper.

SCSS for CNF. We use pPRGs to construct SCSS for various computational representations. Our first construction is a SCSS for monotone CNF formulas.

Theorem 1.4 (SCSS for CNF, informal)**.** *Assume that there is a robust pPRG in which the size of the projective keys is* $\text{poly}(\log m)$*, where* m *is the output length of the generator. Then, there is a SCSS for monotone CNF formulas with share size* $\text{poly}(\log m)$*, where* m *is the number of clauses in the CNF formula.*

The idea for this construction is to start with the information-theoretic secret-sharing scheme of [ISN87] for CNF formulas and use pseudorandom bits in it instead of random bits. In particular, in the construction of [ISN87] for a formula $\varphi(y_1, \ldots, y_n)$ with m clauses, there is a random bit for each clause of the formula and one public bit — the exclusive-OR of the m bits and the secret. The share sh_i contains all bits of clauses that contain y_i. Given a satisfying assignment $x \in \{0,1\}^n$, in each clause there is at least one variable that is satisfied by x, meaning that the shares $(\mathsf{sh}_i)_{i:x_i=1}$ contain the m bits of the clauses, and therefore the secret can be recovered from them and the public bit. Note that in the above scheme, there are m random bits and each share contains a subset of them. This is exactly the functionality provided by a pPRG — the m bits will be the output of the pPRG and the share of a party is the projective key for the appropriate set. In particular, the size of the shares is determined by the length of the projective keys.

As we can represent every monotone function $f : \{0,1\}^n \to \{0,1\}$ by a monotone CNF formula with $m \leq 2^n$ clauses, we can construct a SCSS for all access structures with share size $\mathrm{polylog}(m) = \mathrm{poly}(n)$, i.e., derive Theorem 1.1. In this case, we need the sub-exponential RSA assumption, or a sub-exponentially secure IO scheme and an SSB hash function family.

SCSS for monotone circuits. We generalize the construction for monotone CNF to monotone circuits. Yao [Yao89, VNS+03] showed how to construct a computational secret sharing (CSS) scheme for monotone circuits over OR and AND gates of unbounded fan-in (hereafter referred to as *AND-OR circuits*) with small share size and public information whose size is linear in the number of wires in the circuit (and the security parameter). We use a pPRG to get a construction whose public information size is linear in the number of *gates* which, for general circuits, yields a *quadratic* improvement. In fact, we can get OR gates "for free" and pay only for AND gates (this yields a strict generalization of the CNF construction).

Theorem 1.5 (Quadratic saving for efficient functions, Informal). *Assume that there is a robust pPRG in which the length of the projective keys is* $\mathrm{polylog}(m)$*, where* m *is the output length of the generator. Then, there is a SCSS for monotone AND-OR circuits with share size* $\mathrm{polylog}(m)$ *and public information of length* $m_\wedge$*, where* m *is the number of gates and* $m_\wedge$ *is the number of AND-gates.*

SCSS for graphs. Let us move back to "low-end" functions and consider secret-sharing schemes for graphs. An undirected graph $G = (V = \{1,\ldots,n\}, E)$ represents the monotone function $f : \{0,1\}^n \to \{0,1\}$ whose minterms are the edges of G. That is, a coalition I should be able to recover the secret if and only if I contains a pair of parties, $i, j \in [n]$ such that $(i,j) \in E$. Equivalently, any such graph structure can be represented by a monotone 2-DNF. Graph access structures form an intriguing example of a simple family of functions for which we can hardly get any improvement over the trivial upper bound of $O(n)$ on the per-party share size. Indeed, despite an intensive study (e.g., [BD91, BS92, CSGV93, vD95, BDDV97, Csi05, CT13, Csi09, Csi15, BFM16, FKMP18, BF20]), the best-known information-theoretic secret-sharing scheme for graphs have per-party share size $O(n/\log n)$ [EP97]. In contrast, we construct a SCSS for graphs with $\mathrm{polylog}(n))$ per-party share size.

Theorem 1.6 (SCSS for graphs, Informal). *Assume that there is a robust pPRG in which the length of the projective key is* $\mathrm{polylog}(m)$*, where* m *is the output length of the generator. Then, there is a SCSS for graphs with per-party share size* $\mathrm{polylog}(n)$*, where* n *is the number of vertices in the graph. In particular, under the sub-exponential RSA assumption, there is a SCSS for graphs with share size* $\mathrm{polylog}(n)$*.*

The idea of this scheme is to start with the following simple information-theoretic scheme: Let $r_1, \ldots, r_n$ be random bits. The share sh_i contains $r_i \oplus s$ and $(r_j)_{(i,j)\in E}$. For $(i,j) \in E$, the share sh_i contain r_j and the share sh_j contains $r_j \oplus s$, thus the secret s can be reconstructed from these two shares. In this scheme, all bits (but one) held by each party are random bits, thus, we can use a pPRG to generate them. The share sh_i will contain the bit $r_i \oplus s$ and the projective key for the (pseudo)-random bits corresponding to the set of its neighbors.

1.1.2 SCSS from One-Way Functions

For several classes of functions, we can construct SCSS schemes under the most basic cryptographic assumption that one-way functions exist. We first consider a central sub-class of monotone functions that we call *partite* functions.

Partite Functions (aka CDS). Partite functions $f : \{0,1\}^{2n} \to \{0,1\}$ are monotone functions that encode every (possibly non-monotone) function $g : \{0,1\}^n \to \{0,1\}$ by essentially allocating a variable to each original literal, i.e., $f(x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n) = g(x_1, \ldots, x_n)$ for every $x \in \{0,1\}^n$. (See Definition 5.1 for a formal definition.) Secret sharing for partite functions is an extremely useful primitive that captures cases where the parties are partitioned into pairs, and we mainly care about coalitions that contain exactly one party from each pair. For example, this notion was studied by [BI05, VV15] under the terminology of *non-monotone secret sharing* schemes. Furthermore, secret-sharing schemes for partite functions are equivalent to *multi-server fully-decomposable conditional disclosure of secrets* (CDS) protocols, a cryptographic primitive introduced by Gertner et al. [GIKM00], which has interesting applications, such as symmetric private information retrieval protocols [GIKM00], attribute based encryption [GKW15, Att14, Wee14, IW14], (priced) oblivious transfer [AIR01], and secret-sharing schemes [LV18, ABF$^+$19, ABNP20, AN21]. Finally, partite secret-sharing schemes are essentially equivalent to *privacy-free garbling schemes* [FNO15], and can also be captured by *partial garbling schemes* [IW14]. The best-known share size in (information-theoretic) secret-sharing schemes for general partite functions is $2^{O(\sqrt{n}\log n)}$ [LVW18].

Theorem 1.7 (SCSS for partite functions, informal)**.** *Assuming one-way functions with sub-exponential security exist, for partite functions $f : \{0,1\}^{2n} \to \{0,1\}$, represented by truth tables of size $N = 2^n$, there exists a SCSS with share size $\mathrm{polylog}(N) = \mathrm{poly}(n)$.*

As before, under the existence of polynomially-hard OWF, we can achieve share size N^{ε} for any constant $\varepsilon > 0$. The same applies to the other results in this subsection.

Thinking of secret-sharing for general partite functions as a weak form of "non-monotone" secret-sharing, we get a high-end result ("every function can be realized by a non-monotone SCSS with polylogarithmic complexity") that is weaker than our strongest RSA-based theorem (Theorem 1.1) but can be based on a more conservative cryptographic assumption.

Forbidden graphs. Secret-sharing schemes for *forbidden graphs* [SS97] are similar to secret sharing for graphs discussed above, except that all triplets are authorized. That is, an undirected graph $G = (V = \{1, \ldots, n\}, E)$ defines a function $f : \{0,1\}^n \to \{0,1\}$ that accepts an input I, representing a subset of $[n]$, if $|I| \geq 3$, or if I is an edge in the graph. That is, f is a *slice* function that rejects (resp. accepts) inputs of weight smaller (resp., larger) than 2, and handles weight-2 inputs according to the specification of the graph G. Secret-sharing schemes for forbidden graphs are equivalent (up to a factor of $\log n$) to 2-server conditional disclosure of secrets (CDS) protocols, defined in [GIKM00]. The following theorem can be obtained as a corollary of Theorem 1.7.

Theorem 1.8 (SCSS for forbidden graphs from one-way functions, informal). *Assuming one-way functions with sub-exponential security exist, for every n-vertex graph G there exists a SCSS for the forbidden graph G with share size* $\mathrm{polylog}(n)$.

Secret sharing for forbidden graphs seems easier to realize than for graphs: One can easily turn a general graph SCSS into a forbidden-graph SCCS with a small overhead, but no such transformation is known in the converse direction. Thus, compared to Theorem 1.6, Theorem 1.8 applies to a more restricted family of functions but employs a weaker assumption.

SCSS for DNFs. We move on from sub-classes of monotone functions to general monotone functions represented by monotone DNF formulas. The best information-theoretic secret-sharing for monotone DNF formulas [ISN87] has share size m, the number of terms in the formula. This scheme can be generically transformed into a computational secret-sharing scheme with small shares and public information of size $O(mn)$ (place an encryption of each share in the public file and hand each party the corresponding key). We show that one can do better, obtaining SCSS for monotone DNF with small share size and public information of size only m.

Theorem 1.9 (SCSS for DNFs from one-way functions). *Assuming one-way functions with sub-exponential security exist, there exists a SCSS for monotone DNF formulas over n variables and $m > n$ terms with* $\mathrm{polylog}(m)$ *share size and m bits of public information.*

We show that for some monotone DNF formulas, the public information can be eliminated, and use the scheme to derive the first separation between computational and information-theoretic single-bit secret-sharing. Our separation strongly relies on the best-known lower bound of $\Omega(n^2/\log n)$ on the share size in information-theoretic secret-sharing scheme of [Csi97, Csi96]. We tweak the original function from [Csi96] in a way that keeps the $\Omega(n^2/\log n)$ information-theoretic lower-bound valid but makes the resulting function simple enough to be realized by a DNF-based SCSS of share size $\tilde{O}(n)$.

Theorem 1.10 (Separation). *There exists an infinite sequence of functions $(f_n : \{0,1\}^n \to \{0,1\})_{n\in\mathbb{N}}$ such that, assuming that one-way functions exist, there is a SCSS for these f_n with* total *share size $\tilde{O}(n)$. In contrast, in every information-theoretic secret-sharing scheme for f_n the total share size is $\Omega(n^2/\log n)$.*

We summarize our results in Table 1.

1.1.3 Discussion: SCSS and pPRG vs. Other Primitives

The abstraction of projective PRGs that we introduce in this work seems on the face of it closely related to several objects studied in "mid-range" cryptography, some prominent examples include broadcast encryption [FN94, BGW05], laconic oblivious transfer [CDG+17], trapdoor hash functions [DGI+19], laconic secure function evaluation [QWW18], and attribute-based encryption (ABE) [SW05, GPSW06]. These primitives can be constructed from by-now standard cryptographic assumptions such as LWE or Diffie-Hellman-like assumptions on groups that support bilinear maps. Yet, there are significant differences.

For example, consider a weak version of broadcast encryption, namely, a secret-key (single-ciphertext-secure) broadcast encryption versus projective PRGs. In both cases, there is an algorithm that takes in a master secret key and a set T and outputs a succinct key (of size much shorter than $|T|$). On the other hand, in broadcast encryption, there is a single encrypted bit which can be recovered using additional user-specific keys. In a pPRG, there are m "encrypted bits" and recovering the appropriate subset can be done using just the projected key (and no additional information), a seemingly harder task. Using stronger forms of broadcast encryption such as many-key, many-ciphertext-secure secret-key broadcast encryption, or even public-key broadcast encryption, do not seem to help either. Finally, using a broadcast encryption scheme to construct SCSS runs into trouble as well: a ciphertext in a broadcast encryption scheme is associated with a single set S whereas in a secret-sharing scheme, we seemingly need to compress ciphertexts associated to all the (exponentially many) authorized sets into a succinct representation. Nonetheless, we can use an appropriately succinct broadcast encryption scheme to construct non-trivial SCSS schemes with shares of size $\tilde{O}(2^{n/2})$ as described below.

The relation to ABE. Let us move on to the case of symmetric-key ABE which is closer in spirit to secret sharing. Syntactically, the ABE key-generation algorithm takes a predicate $f : X \times Y \to \{0,1\}$ and generates, for every policy $x \in X$, decryption key dk_x, as well as a private "global" encryption key ek that is not attached to x. (For symmetric-key ABE, we can assume, w.l.o.g., that the encryption key is taken to be the random coins of the key-generation algorithm.) Encryption uses ek to map a message and an attribute y to a ciphertext c, and decryption over dk_x recovers the message if $f(x,y) = 1$. Consider the following, somewhat-informal, security notions given in increasing order (from weak to strong). In all cases, the ciphertext does not hide the attribute string y.

1. (single message given single key): An adversary who chooses a single policy x, and gets the decryption key dk_x, "learns nothing" on a single message that is encrypted under an attribute y for which $f(x,y) = 0$. This notion is essentially equivalent to so-called forbidden graph secret sharing over bipartite graphs (aka 2-party CDS). Roughly, a bipartite graph over $X \times Y$ can be naturally associated with a predicate $f : X \times Y \to \{0,1\}$, and the share of a left party x (resp., right party y) is associated with the decryption key dk_x (resp., with the ciphertext that encrypts the secret s under the attribute y). By Theorem 1.8, such ABE can be constructed based on one-way functions.

2. (single message given multiple keys): An adversary who chooses many policies $x_1, \ldots, x_k$, and gets the corresponding decryption keys $\mathsf{dk}_{x_1}, \ldots, \mathsf{dk}_{x_k}$, "learns nothing" on a single message that is encrypted under an attribute y for which $f(x_1,y) = \ldots = f(x_k,y) = 0$. This notion is equivalent to *graph secret sharing* over bipartite graphs (aka fully-robust 2-party CDS [ABNP20]) via the previous transformation. A succinct solution to this problem for general predicates $f : [N] \times [N] \to \{0,1\}$ with complexity of $\mathrm{poly}(\log N, \lambda)$ suffices for getting general n-party secret sharing with complexity of $\tilde{O}(2^{n/2})$ via a standard reduction [Pet20, ABNP20, Bei23]. Since such succinct ABE schemes exist from the (subexponential) LWE assumption [BGG+14] and hence, we get a general n-party secret sharing with complexity of

$\tilde{O}(2^{n/2})$ from the same assumption. We describe this construction in more detail at the end of this section.

3. (*ciphertext-free* ABE for pseudorandom messages with security against multiple keys): In contrast, a pPRG scheme yields a stronger form of succinctness: Each attribute y is associated with a single pseudorandom bit b_y that is induced by the key-generation algorithm, and can be recovered, given a matching decryption key dk_x for which $f(x, y) = 1$, without sending *any additional ciphertext*! Furthermore, even if one holds many keys for different policies, all the bits that are associated with attributes that are not authorized, cannot be recovered (remain pseudorandom). This extra feature allows us to realize CNFs succinctly.[3]

Note that LWE yields ABE with stronger security properties, such as the security of multiple encryption queries (which is trivially true in the public-key setting) given multiple keys. However, it is not clear whether these extra features can be converted into a ciphertext-free property. It should also be noted that, while pPRGs seem like a strong object, we do not even know whether public-key assumptions are necessary for realizing them.

As described in bullet (2) above, the existence of secret-key attribute-based encryption (ABE) schemes with succinct keys or ciphertexts implies n-party SCSS schemes with share size roughly $\sqrt{N} = 2^{n/2}$ which, while a far cry from the polylogarithmic share size offered by Theorem 1.1, is still an improvement over the best known information-theoretic constructions. Here, succinctness refers to keys (or ciphertexts) with size that grows polylogarithmically with the size of the policy circuit (or the attributes). Such ABE schemes are known from the LWE assumption [BGG+14] and from the bilinear Diffie-Hellman exponent assumption [BGW05].[4]

For the sake of completeness, let us directly describe how to construct SCSS based on succinct ABE schemes. (This is essentially the reduction from [Pet20, ABNP20, Bei23].) Consider a symmetric-key ABE scheme as in bullet (2) above. To share a bit b, do the following. For every string $x_i \in \{0,1\}^{n/2}$, generate a secret key dk_{x_i} and for every $y_j \in \{0,1\}^{n/2}$, generate a ciphertext ct_{y_j} encrypting the secret bit b. Then:

- Publish $\mathsf{dk}_{0^{n/2}}$ and $\mathsf{ct}_{0^{n/2}}$.
- Additively secret-share dk_x, for each $x \in \{0,1\}^{n/2} \setminus \{0^{n/2}\}$, among all the players $1 \le i \le n/2$ for which $x[i] = 1$.
- Additively secret-share ct_y, for each $y \in \{0,1\}^{n/2} \setminus \{0^{n/2}\}$, among all the players $n/2+1 \le i \le n$ for which $y[i] = 1$.

If players in a set (whose characteristic vector is) $x||y$ come together, they can recover dk_x and ct_y and therefore the secret bit b if and only if $f(x, y) = 1$.

[3]Indeed, if we relax security and assume that the adversary holds only a single key, this notion is essentially equivalent to puncturable PRFs [KPTZ13, BW13, BGI14] that can be based on (sub-exponentially strong) one-way functions.

[4]The latter construction is via the construction of broadcast encryption with constant-size secret keys and ciphertexts and linear-size public parameters from [BGW05] which immediately gives a succinct ABE scheme of the type we need. See also [Wee16, CMM16] for constructions based on simpler assumptions related to bilinear maps.

1.2 Other Related Works

For uniform access structures that are succinctly described by a Turing Machine (TM), the witness encryption based approach of [KNY17] can be made succinct (in the sense of having shorter shares than the TM runtime) by applying witness encryption for TMs, which in turn follows from iO for TMs or just standard (subexponential) iO [BCG$^+$18, KLW15, IPS15]. However, for general access structures which are not decidable by succinct Turing machines, these results do not give us SCSS schemes.

The use of computational assumptions in secret sharing is not new. Krawczyk [Kra94] considered sharing *large* secrets supporting threshold access structures and showed how to use computational assumptions to obtain shares of size smaller than the secret, breaking an information-theoretic lower-bound proved by Karnin, Greene, and Hellman [KGH83]. Yao [Yao89, VNS$^+$03] showed how to construct, assuming the existence of one-way functions, a computational secret-sharing scheme with polynomial-size shares for functions f computable by $\mathrm{poly}(n)$-size monotone circuits, a feat that is known in the information-theoretic world only for the special case of *formulas* [BL90]. Komargodski, Naor, and Yogev [KNY17] gave a construction of a computational secret-sharing scheme for any monotone function in NP (assuming witness encryption for NP and one-way functions). The shares in Yao's scheme and in Komargodski et al.'s scheme grow with the circuit size, and so these schemes do not seem to achieve succinctness in our sense. Cachin [Cac95] constructed a computational secret-sharing scheme for arbitrary monotone functions, where the size of each share is the size of the secret, however, it has long public information of size 2^n.

Computational assumptions were widely used in order to support security under active adversaries starting with the works of Feldman [Fel87] and Pedersen [Ped91] on *computational verifiable secret sharing*, and also in the more restricted setting of *robust* secret sharing [Kra94]. A formal treatment of computational secret-sharing and robust secret-sharing was given by Bellare and Rogaway [BR07].

Organization. Following some preliminaries (Section 2), we define and construct projective PRGs (Section 3) and use them to derive SCSS for graphs, CNF formulas, truth tables and monotone circuits (Section 4). Finally, based on one-way functions, we construct SCSS for Partite Functions and Forbidden Graphs (Section 5) and present a separation between computational secret sharing and information-theoretic secret sharing (Section 6).

2 Preliminaries

Notation. Let $[m]$ denote the set $\{1, 2, \ldots, m\}$. For a string $x \in \{0,1\}^n$ and an index $i \in \{0,1\}^n$, let $x[i]$ denote the i^{th} bit of x, and for a subset $S \subseteq [n]$, let $x[S]$ denote the substring of x indexed by the locations in S. For a finite set S, let $x \leftarrow_U S$ denote sampling x uniformly at random from S. For a string $x \in \{0,1\}^n$, we denote $S_x = \{i : x_i = 1\}$. A function $\varepsilon(n)$ is a negligible function if for every positive polynomial function $p(n)$, there exists $n_0 \in \mathbb{N}$ s.t. $f(n) < 1/p(n)$ for all $n > n_0$. For two distributions D_1 and D_2, let $D_1 \approx_c D_2$ denote the fact that D_1 and D_2 are computationally indistinguishable.

Definition 2.1 (Monotone functions and minterms). *For* $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in$

$\{0,1\}^n$, we write $x \leq y$ if $x_i \leq y_i$ for every $i \in [n]$, and we write $x < y$ if $x \leq y$ and $x \neq y$. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is monotone *if $f(x) \leq f(y)$ for every $x \leq y$. A minterm of a monotone function f is an input $y \in \{0,1\}^n$ such that $f(y) = 1$ and $f(x) = 0$ for every $x < y$.*

One-way functions and output-variable pseudorandom generators. A $\mathrm{poly}(\lambda)$-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is $t(\lambda)$-secure one-way function (OWF) if for every $t(\lambda)$-time non-uniform adversary $\mathcal{A}$ and all sufficiently large λ, it holds that

$$\Pr_{x \in_U \{0,1\}^\lambda}[\mathcal{A}(1^\lambda, f(x)) \in f^{-1}(f(x))] \leq 1/t(\lambda).$$

If f is a $t(\lambda)$-secure OWF for every polynomial $t(\lambda)$ then is a *polynomially-secure* OWF, or simply OWF. If f is a $\exp(\lambda^\delta)$-secure one-way function for some constant $\delta > 0$ then it is a *sub-exponentially-secure* OWF. A probability distribution D over m-bit strings is t-pseudorandom if for every t-size circuit $\mathcal{A}$ it holds that

$$\left| \Pr_{x \in_U D}[\mathcal{A}(D) = 1] - \Pr_{y \in_U \{0,1\}^m}[\mathcal{A}(y) = 1] \right| \leq 1/t.$$

We will need the following somewhat non-standard notion of an output-variable pseudorandom generator (PRG) as defined in [Gol01, Def. 3.3.4]. A $t(\lambda)$-secure PRG is a polynomial-time algorithm G that takes a seed $a \in \{0,1\}^\lambda$ and an output length parameter 1^m and outputs a string r of length m such that for all sufficiently large λ, the distribution $G(a, 1^{t(\lambda)})$, induced by $a \in_U \{0,1\}^\lambda$, is $t(\lambda)$-pseudorandom.[5] When the output length 1^m is clear from the context we will omit it and treat G as a mapping from λ bits to m bits. We say that G is a *polynomially-secure* PRG, or simply PRG, if it is $t(\lambda)$-secure PRG for every polynomial $t(\lambda)$. If G is $\exp(\lambda^\delta)$-secure PRG for some constant $\delta > 0$ then it is a *sub-exponentially-secure* PRG. By the classical result of [HILL99] and standard stretch-expanding techniques (e.g., [GGM86]), for every function $t(\lambda) > \lambda$ the existence of $t(\lambda)$-secure OWF implies the existence of $t'(\lambda)$-secure PRG where $t' = t^c$ for some universal constant c. In particular, polynomially-secure (resp. sub-exponentially secure) one-way functions imply polynomially-secure (resp. sub-exponentially secure) PRGs.

2.1 Computational Secret-Sharing Schemes

In this section we define *computational* secret-sharing schemes, that is, secret-sharing schemes in which the sharing and reconstruction are efficient, and in which no polynomial-time adversary can learn any information about the secret from the shares of any unauthorized set of parties. When defining "efficiency" it is important to consider the way the access structure is represented. In this paper, we will represent functions in several ways, e.g., as CNF formulas, DNF formulas, monotone circuits, non-monotone circuits, and truth tables. To unify the different results, we start with an abstract definition of a representation model for Boolean functions.

[5] In [Gol01, Def. 3.3.4] it is also required that for every seed $a \in \{0,1\}^\lambda$ and integer t it holds that $G(a, 1^t)$ is a prefix of $G(a, 1^{t+1})$. We can adopt this convention as well although it is not crucial for us.

Definition 2.2 (Representation model). *A* representation model *is a polynomial time computable function $U : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, where $U(P,x)$ is referred to as the value returned by a "program" P on an input x. We assume that each P specifies an input length n and $|P| \geq n$. We say that P represents the function $f : \{0,1\}^n \to \{0,1\}$ in the representation model U if $f(x) = U(P,x)$ for every $x \in \{0,1\}^n$.*

The correctness and security goals of secret sharing are defined with respect to a collection of authorized sets called an *access structure*.

Definition 2.3 (Authorized sets and access structures). *We consider a set of n parties $\{1,\ldots,n\}$ and think of any string $x \in \{0,1\}^n$ as representing the set of parties $S_x = \{i : x_i = 1\}$. Furthermore, we think of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ as representing a collection of subsets of the parties $\{1,\ldots,n\}$, called an* access structure*, consisting of all sets S_x such that $f(x) = 1$. We refer to such a set of parties S_x as being* authorized.

In this work, we will only be interested in *monotone* access structures f. We will typically consider representation models U that are *universal* in the sense that for every monotone $f : \{0,1\}^n \to \{0,1\}$ there is a program P representing f.

We now define our main notion of computational secret sharing. For simplicity, we assume that the secret is a single bit. This can always be generalized to sharing an ℓ-bit secret by independently sharing each bit of the secret. Moreover, using a symmetric encryption scheme, the amortized cost of sharing long secrets can often be improved (see Remark 2.9).

Definition 2.4 (Computational secret sharing – Syntax and correctness). *A computational secret-sharing (CSS) scheme for a representation model U consists of a pair of algorithms* $\mathsf{CSS} = (\mathsf{CSS}.\textsc{Share}, \mathsf{CSS}.\textsc{Recon})$ *with the following syntax.*

Sharing. $\mathsf{CSS}.\textsc{Share}(1^\lambda, P, s) \to (\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ *(where n denotes the input length of P) is a randomized poly-time algorithm that takes as input a security parameter λ, a program P, and a secret $s \in \{0,1\}$ and outputs $n+1$ shares $\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n$, where sh_i, for $1 \leq i \leq n$, is the share of the i^{th} party and sh_0 is a public information given to all parties. By default, the public information sh_0 is an empty string.*

Reconstruction. $\mathsf{CSS}.\textsc{Recon}(P, x, \mathsf{sh}_0, (\mathsf{sh}_i)_{i \in S_x}) \to s$ *is a deterministic poly-time algorithm that takes as input a program P, a string $x \in \{0,1\}^n$ describing the reconstructing set (where n denotes the input length of P), a public information sh_0, and shares of the parties in S_x. The algorithm outputs a secret $s \in \{0,1\}$.*

We say that CSS *is* correct *(with respect to U) if for every λ, s, program P, and input $x \in \{0,1\}^n$ such that $P(x) = 1$ (where n denotes the input length of P), the process of invoking* $\mathsf{CSS}.\textsc{Share}(1^\lambda, P, s)$ *and then invoking* $\mathsf{CSS}.\textsc{Recon}(P, x, \mathsf{sh}_0, (\mathsf{sh}_i)_{i \in S_x})$ *(with the sh_i generated by* $\mathsf{CSS}.\textsc{Share}$*) always returns s.*

Share size. When discussing the share size of computational secret-sharing schemes, we will consider two measures: the (maximal) *share size* $\max\{|\mathsf{sh}_i| : i \in [n]\}$ and the *public information size* $|\mathsf{sh}_0|$ (0 by default). We would like the share size to be very small, ideally polylogarithmic in the size of the representation and polynomial in λ.

Definition 2.5 (Computational secret sharing – Security). *Consider the following game between a non-uniform $t(\lambda)$-time adversary $\mathcal{A}$ and a challenger:*

1. *The adversary $\mathcal{A}$ on input 1^λ chooses P and an input $x \in \{0,1\}^n$ such that $P(x) = 0$ (where n is the input length of P) and sends them to the challenger.*
2. *The challenger chooses a secret $s \leftarrow_U \{0,1\}$ uniformly at random. It computes $(\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathrm{CSS}.\textsc{Share}(1^\lambda, P, s)$ and sends $\mathsf{sh}_0, (\mathsf{sh}_i)_{i \in S_x}$ to the adversary.*
3. *The adversary outputs a bit s'.*

The adversary wins the game if $s' = s$.

We say that CSS *is* $t(\lambda)$-secure *if for every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$ and sufficiently large λ, the probability that $\mathcal{A}$ wins is at most $1/2 + 1/t(\lambda)$. By default, we require $t(\lambda)$-security for every polynomial $t(\cdot)$. In any case, we always assume that $t > \lambda$.*

Remark 2.6 (Alternative single-parameter definition). *Definition 2.5 uses the standard convention of allowing the adversary's resources to depend only on the security parameter λ. This should be contrasted with the simpler single-parameter definition used in the introduction, where the size of P is also used as a security parameter, and any adversary of size $\mathrm{poly}(|P|)$ should have a negligible advantage. Under sub-exponential security assumptions (resp., standard polynomial assumptions) we can use $\lambda = \mathrm{polylog}(|P|)$ (resp., $\lambda = O(|P|^\varepsilon)$ for every $\varepsilon > 0$) to obtain a scheme realizing the single-parameter definition from a scheme realizing Definition 2.5.*

Remark 2.7 (Information-theoretic secret sharing). *The standard notion of (perfect) information-theoretic secret sharing can be defined similarly by a pair* $\mathrm{ITSS} = (\mathrm{ITSS}.\textsc{Share}, \mathrm{ITSS}.\textsc{Recon})$ *with the following differences: the algorithms* $\mathrm{ITSS}.\textsc{Share}$, $\mathrm{ITSS}.\textsc{Recon}$, *and $\mathcal{A}$ are computationally unbounded, and $\varepsilon(\lambda) = 0$ (so that the input λ can be eliminated). Furthermore, in this setting the public information sh_0 is not useful and can also be eliminated.*

Remark 2.8 (Trading share size for public information size). *In computational secret sharing, it is possible to generically shrink the private share size by using public information. Concretely, if we have a CSS scheme producing shares $\mathsf{sh}_1, \ldots, \mathsf{sh}_n$, then the dealer can encrypt the share sh_i of party p_i with a semantically-secure symmetric encryption scheme using an independent key k_i; the share of p_i is k_i and the public information is the encryption of the n original shares. Thus, the size of each share is λ (the key length) and the length of the public information is the total size of the shares in the original scheme. Thus, this transformation does not decrease the total size of all shares when we include the public information.*

Remark 2.9 (Dispersing public information [Kra94]). *The above transformation will typically result in a large amount of public information. If all authorized sets are big, say of size n/c for some small constant c, the public information can be efficiently dispersed between the parties by using any good erasure code (such as a Reed-Solomon code), as observed by Krawczyk [Kra94]. This results in a CSS scheme with no public information, where the* total *size of all shares is comparable to the total size of all information (including public information) in the original scheme. As a corollary, for such an access structure, a secret of size ℓ can be shared with information rate that tends to $1/c$ when ℓ tends to infinity (assuming the existence of a one-way function). However, this asymptotic rate may only "kick in" when $\ell = \Omega(2^n)$. In this work, we focus on minimizing the share size for 1-bit secrets.*

Definition 2.10 (Partial access structures). *A partial function $f : \{0,1\}^n \to \{0,1,\perp\}$ is* monotone *if there is no pair of inputs x, x' for which $x \le x'$, $f(x) = 1$, and $f(x') = 0$. All the above definitions naturally generalize to the case of partial monotone functions. Specifically, correctness should hold for every input x that is accepted by f (i.e., $f(x) = 1$) and privacy should hold for every input x that is rejected by f, i.e., $f(x) = 0$. The scheme may act arbitrarily over other inputs x for which $f(x) = \perp$ (e.g., the parties in S_x can learn partial information on the secret).*

3 Projective Pseudorandom Generators

An important conceptual contribution of this paper is introducing a new cryptographic primitive called projective pseudorandom generators (pPRGs). We start this section by defining pPRGs and then show several constructions: from the RSA assumption in Section 3.2; from iO and SSB hash functions (Section 3.3); from Diffie-Hellman-like assumptions in Section 3.4; and from the learning with errors assumption in Appendix A.

3.1 Definition of Projective PRGs

Intuitively, a projective pseudorandom generator (pPRG) is a PRG $G : \{0,1\}^\lambda \to \{0,1\}^m$ with an additional property. Given a master key and a subset $T \subseteq [m]$ of the output indices, one can produce a projective key $a\{T\}$, which can be used to recover the subset of output bits of G indexed by the set T, namely $G(a)[T]$, but reveals nothing about the other bits. This by itself is trivial as $a\{T\}$ could simply be $G(a)[T]$. The feature that makes the notion usefull and non-trivial is a succinctness requirement on the projected seed, that is, the bit-length of $a\{T\}$ should be significantly smaller than $|T|$. We define *weak succinctness* where $|a\{T\}| = O(|T|^{1-\delta})$ for some constant $\delta > 0$, and *strong succinctness* where $|a\{T\}| = (\log |T|)^{O(1)}$. For technical reasons, we assume that the output length of the pPRG is given as an additional input. Also, for our purposes, the seed may be long and may be sampled from some arbitrary distribution; we therefore refer to it as a *master secret* key msk.

Definition 3.1. *A* projective PRG *(pPRG) is a triple of algorithms* $\text{pPRG} = (\text{pPRG.}\textsc{Setup}, \text{pPRG.}\textsc{KeyGen}, \text{pPRG.}\textsc{Eval})$ *with the following syntax.*

Setup. $\text{pPRG.}\textsc{Setup}(1^\lambda, 1^m) \to (\mathsf{params}, \mathsf{msk})$ *is a randomized poly-time algorithm that takes as input a security parameter λ and an output length parameter m, and samples public parameters* params *and master secret key* msk*. We assume that the public parameters are of length* at least λ *and that one can recover in time* $\text{poly}(\lambda, \log m)$ *the values of the security parameter and the output length m from* params*.*

Key Generation. $\text{pPRG.KeyGen}(\mathsf{params}, \mathsf{msk}, T)$ *is a deterministic poly-time algorithm that takes as input the public parameters* params*, a secret key* msk*, and a set* $T \subseteq [m]$ *represented by its* m*-bit characteristic vector, and outputs a projective key* $a\{T\}$*.*

Evaluation. $\text{pPRG.Eval}(\mathsf{params}, a\{T\}, T)$ *is a deterministic poly-time algorithm that takes as input the public parameters* params*, a projective key* $a\{T\}$*, and a set* $T \subseteq [m]$ *represented by an* m*-bit characteristic vector, and outputs a string* $y \in \{0,1\}^{|T|}$*. We slightly abuse notation and let* $\text{pPRG.Eval}(\mathsf{params}, \mathsf{msk})$ *denote the outcome* $c \in \{0,1\}^m$ *of* $\text{pPRG.Eval}(\mathsf{params}, a\{\mathsf{all}\}, [m])$*, where* $a\{\mathsf{all}\} = \text{pPRG.KeyGen}(\mathsf{params}, \mathsf{msk}, [m])$ *is the projective key that corresponds to the entire set of outputs. We refer to* c *as the string that is generated by* $(\mathsf{params}, \mathsf{msk})$*, or simply as the output of the pPRG.*

Since the description length of params *(resp.,* T*) is at least* λ *(resp., the output length* m*), the algorithms* pPRG.KeyGen *and* pPRG.Eval *are implicitly allowed to run in time* $\text{poly}(\lambda, m)$*. We require the following properties:*

Correctness. *Correctness requires that for every* λ, m*,* $(\mathsf{params}, \mathsf{msk}) \in \text{pPRG.Setup}(1^\lambda, 1^m)$*,* $T \subset [m]$*, and* $a\{T\} = \text{pPRG.KeyGen}(\mathsf{params}, \mathsf{msk}, T)$*, it holds that*

$$y = c[T],$$

where $y = \text{pPRG.Eval}(\mathsf{params}, a\{T\}, T)$ *is the string that is generated by the* T*-projective key* $a\{T\}$ *and* $c = \text{pPRG.Eval}(\mathsf{params}, \mathsf{msk})$ *is the string generated by* $(\mathsf{params}, \mathsf{msk})$*.*

Succinctness. *Weak succinctness requires that there is a constant* $\delta > 0$ *such that for every* $T \subseteq [m]$ *the bit-length of* $a\{T\}$ *is* $m^{1-\delta}\,\text{poly}(\lambda)$*. Strong succinctness requires that there is a fixed polynomial* p *such that the size of* $a\{T\}$ *is* $p(\log m, \lambda)$*.*

Security. *Consider the following game between an adversary* $\mathcal{A}$ *and a challenger* CA*:*

1. *Given an input* 1^λ*, the adversary* $\mathcal{A}$ *chooses* 1^m *and* $T \subset [m]$ *and sends* $(1^m, T)$ *to the challenger.*
2. CA *samples* $(\mathsf{params}, \mathsf{msk}) \leftarrow \text{pPRG.Setup}(1^\lambda, 1^m)$*, computes* $c \leftarrow \text{pPRG.Eval}(\mathsf{params}, \mathsf{msk})$*, sets* $c_1 \leftarrow c[\overline{T}]$*, and samples* $c_0 \in_U \{0,1\}^{|\overline{T}|}$*, where* $\overline{T} = [m] \setminus T$ *is the complement of* T*. The challenger samples* $b \in_U \{0,1\}$ *and sends* params*,* $a\{T\} = \text{pPRG.KeyGen}(\mathsf{params}, \mathsf{msk}, T)$*, and* c_b *to the adversary.*
3. *The adversary outputs a bit* b' *and wins if* $b = b'$*.*

We say that pPRG *is* $t(\lambda)$-secure *if for every non-uniform* $t(\lambda)$*-time adversary* $\mathcal{A}$ *the probability that* $\mathcal{A}$ *wins is at most* $1/2 + 1/t(\lambda)$*.*

An additional feature of a projective PRG is robustness, a strengthening of the above security definition.

Robustness. *Consider the following game between an adversary* $\mathcal{A}$ *and a challenger* CA*:*

1. *Given an input* 1^λ*, the adversary* $\mathcal{A}$ *chooses* 1^m *and sets* $T_1, \ldots, T_\ell \subset [m]$ *and sends them to the challenger. Let* $T = T_1 \cup \cdots \cup T_\ell$ *denote the union of these sets.*

2. *CA samples* $(\mathsf{params}, \mathsf{msk}) \leftarrow \text{pPRG.Setup}(1^\lambda, 1^m)$*, computes* $c \leftarrow \text{pPRG.Eval}(\mathsf{params}, \mathsf{msk})$*, sets* $c_1 \leftarrow c[\overline{T}]$*, and samples* $c_0 \in_U \{0,1\}^{|\overline{T}|}$*, where* $\overline{T} = [m] \setminus T$ *is the complement of* T*. The challenger samples* $b \in_U \{0,1\}$ *and sends* params*,* $a\{T_i\} = \text{pPRG.KeyGen}(\mathsf{params}, \mathsf{msk}, T_i)$ *for all* $i \in [\ell]$*, and the string* c_b *to the adversary.*
3. *The adversary outputs a bit* b' *and wins if* $b = b'$*.*

We say that pPRG *is* $t(\lambda)$-robust *if for every non-uniform* $t(\lambda)$*-time adversary* $\mathcal{A}$ *the probability that* $\mathcal{A}$ *wins is at most* $1/2 + 1/t(\lambda)$*.*

Remark 3.2 (Defaults). *By default, a pPRG should satisfy* t*-security for every polynomial* $t(\lambda)$*. That is, robustness is viewed as an additional feature. We will also consider sub-exponential adversaries whose running time is* 2^{λ^δ} *for some constant* $\delta > 0$*. Note that in our definition the output length is allowed to scale with the adversary's running time. To highlight this point we will sometimes refer to a polynomially-secure pPRG as having an* arbitrary polynomial *stretch and to a pPRG with sub-exponential security as having a* sub-exponential *stretch.*

Remark 3.3 (PRG from pPRG). *Observe that the mapping* $\text{pPRG.Eval}(\mathsf{params}, a\{\mathsf{all}\}, [m]]) \mapsto y$ *takes a short secret string* $a\{\mathsf{all}\}$ *of length smaller than* m *(and some public strings) and outputs an* m*-bit string such that the output is pseudorandom if the input is sampled from some efficiently samplable distribution, i.e.,* $(\mathsf{params}, \mathsf{msk}) \leftarrow \text{pPRG.Setup}(1^\lambda, 1^m)$ *and* $a\{\mathsf{all}\} = \text{pPRG.KeyGen}(\mathsf{params}, \mathsf{msk}, [m])$*. Indeed, pseudorandomness follows from security in the degenerate case where the adversary chooses* $T = \emptyset$*. Thus any pPRG essentially induces some generalized version of PRG in which the seed,* $a\{\mathsf{all}\}$*, is short but sampled from a non-uniform efficiently samplable distribution. (One can easily transform this into a standard PRG via an elementary use of randomness extractors; details omitted.)*

Remark 3.4 (Bounded robustness). *One can consider a bounded-robustness variant in which the adversary can ask for at most* $\ell(\lambda)$ *keys for some function* ℓ*. This variant, which lies between standard security and full robustness, suffices for some applications. However, since all our constructions achieve full robustness, we will mostly ignore the bounded-robustness variant.*

Remark 3.5 (On the length of the public parameters). *The definition does not put any restriction on the length of the public parameters, and in principle, they can be polynomial in* λ *and the output length. However, as we will later see, long public parameters will be translated to a long public share. It is, therefore, useful to try and compress them to* $\mathrm{poly}(\lambda) \cdot m^{1-\delta}$ *for some constant* $\delta > 0$ *or even to* $\mathrm{poly}(\lambda, \log m)$*. In the latter case, one can get rid of the public parameters by embedding them as part of the projective keys. We can also consider the case of* reusable *public parameters (e.g., it contains a description for a DDH-secure group) that can be re-used with multiple freshly sampled master secret keys* msk*. For this one has to partition the* pPRG.Setup *algorithm into a* params*-sampler and to an* msk*-sampler that receives the sampled public parameters,* params*, as input. By a simple hybrid argument, such a partition guarantees re-usability. Indeed, our bilinear maps-based construction achieves this property.*

3.2 Projective PRG from RSA

We describe a projective PRG based on the RSA assumption using ideas from [AIKW15, ODK+17]. Recall that the RSA assumption asserts that given a random RSA modulus N of bit-length λ, a random exponent $e \leftarrow_U \mathbb{Z}^*_{\varphi(N)}$ and a random element $y \leftarrow_U \mathbb{Z}^*_N$ no $t(\lambda)$-time algorithm can extract the e^{th} root of y except with probability $1/t(\lambda)$. (See Assumption 3.7.) The *sub-exponential RSA assumption* takes $t = 2^{\lambda^\delta}$ for some constant $\delta > 0$, and the *polynomial RSA assumption* asserts that t can be taken to be an arbitrary polynomial. We prove the following theorem.

Theorem 3.6 (Theorem 1.2 restated). *Under the sub-exponential (resp., polynomial) RSA assumption, there exists a sub-exponential-robust pPRG (resp., polynomial-robust pPRG) with sub-exponential stretch (resp., arbitrary polynomial stretch) whose projective keys and public parameters are both strongly succinct, i.e., of length* $\log m \cdot \mathrm{poly}(\lambda)$*, where* m *is the output length. The running time of generating the* m*-bit output of the pPRG is* $\tilde{O}(m) \cdot \mathrm{poly}(\lambda)$*.*

For ease of presentation, we will begin with a pPRG construction, based on [AIKW15], with long public parameters and later explain how to compress them.

Setup. For a security parameter λ and output length m, we let $\mathsf{msk} = (p, q)$, where p and q are random primes of length λ, and publish the following elements as part of params:

$$N = pq, \quad u \leftarrow_U \mathbb{Z}^*_N, \quad r \leftarrow_U \mathbb{Z}^\lambda_2,$$

and a tuple of m random primes, $\vec{e} = (e_1, \ldots, e_m)$, each of length λ.

Key Generation. Given p, q, $\vec{e}$, and a set T we generate a projective key

$$a\{T\} \leftarrow u^{\prod_{j\in T} 1/e_j} \bmod N,$$

where, here and throughout this subsection, all computations in the exponent are computed modulo $\varphi(N) = (p-1)(q-1)$.

Evaluation. Given $\mathsf{params} = (N, u, r, \vec{e})$, a set T, and a projective key $a\{T\}$, the string $(c_i)_{i\in T}$ is generaterated as follows: for every $i \in T$, raise $a\{T\}$ to the power of $\prod_{j\in T\setminus\{i\}} e_j$ record the result as

$$y'_i \leftarrow (a\{T\})^{\prod_{j\in T\setminus\{i\}} e_j} \bmod N,$$

and set $c_i = \mathsf{hc}(r, y'_i)$, where hc is the Goldreich-Levin hard-core bit (i.e., inner product modulo 2).

The full output $(c_1, \ldots, c_m)$ of the pPRG (defined by using the evaluation algorithm with a fully-authorized projective key $a\{\mathsf{all}\}$) is given by $c_i = \mathsf{hc}(y_i, r)$, where $y_i = u^{1/e_i} \bmod N$. Correctness follows from the fact that if $i \in T$,

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

The proof of robustness is given in Section 3.2.2.Before that, we explain how to reduce the size of the public parameters.

3.2.1 Reducing the Size of the Parameters

In the above construction, the projective keys are of length λ but the public parameters, which contain the random primes $\vec{e} = (e_1, \dots, e_m)$ are long; we would like to replace them with some succinct representation of length $\text{poly}(\lambda, \log m)$. The idea is to generate the primes in a "pseudorandom" way $(e_1, \dots, e_m) \leftarrow S(\rho)$, where $S(\rho)$ is an algorithm called sampler, and provide the short seed ρ as part of the public parameters. This should be done with some care since the pPRG adversary gets to see the seed ρ as part of the public parameters. It turns out that the security reduction to RSA of [AIKW15] goes through as long as the sampler S satisfies the following properties: (1) Except with negligible probability, the outputs of S are m distinct primes; (2) The marginal distribution of every single prime is uniform over the set P_λ of primes of length λ; and (3) There exists an efficient *planting* procedure that, given a prime e and an index i, samples a random seed ρ_0 that generates e as its i^{th} output. A sampler that satisfies (1) – (3) is called *admissible*. We present a simple construction of such a sampler with randomness complexity of $O(\lambda^3 \log m)$ and time complexity of $\tilde{O}(m) \cdot \text{poly}(\lambda)$, and modify the above construction in a natural way, i.e., replace $\vec{e}$ with the seed ρ and let the evaluation and key generation compute $\vec{e} \leftarrow S(\rho)$. As a result, the total length of the public parameters is $\lambda^3 \log(m) + O(\lambda)$, and can be even integrated as part of the projective keys(following Remark 3.5).

An admissible sampler S. Assume that m is at most, say, $2^{\lambda/3}$.[6] Let $k = \lambda(\lambda + \log(m))$. The idea is to sample mk integers of bit length λ via some k-wise independent distribution. We partition these integers to m blocks of size k each and let e_i denote the first prime in the i^{th} block. If some block does not contain a prime, we declare a "failure" and, say, set all e_i's to 1. By the density of primes, the probability of this event is at most $m \cdot (1 - 1/\Theta(\lambda))^{\Omega(k)} = \exp(-\Omega(\lambda))$. Also, the probability that we get the same prime twice is upper-bounded by the probability that some integer occurs twice, which is at most $(mk)^2 2^{-\lambda} = \exp(-\Omega(\lambda))$. Furthermore, since the distribution is k-wise independent the residual distribution of each generated prime is uniform (conditioned on not failing). To derive an efficient planting algorithm, let us assume that the sampler S is realized by a polynomial-based k-wise independent distribution. That is, the seed consists of a tuple of k coefficients, $\rho = (\rho_0, \dots, \rho_{k-1})$, sampled uniformly from the field $\mathbb{F}_{2^\lambda}$. The seed specifies a univariate degree-k polynomial $P_\rho(x) = \sum_i \rho_i x^i$ over $\mathbb{F}_{2^\lambda}$ and the j^{th} output of S is the value $P_\rho(\alpha_j)$, viewed as a λ-bit integer, where $\alpha_1, \dots, \alpha_{mk}$ is some canonical set of distinct evaluation points in $\mathbb{F}_{2^\lambda}$. Now efficient planting follows from efficient interpolation. In more detail, given an index i and a prime e, we sample a random block of k integers, place e in the location of the first prime, and find a random seed ρ that generates this modified block as its i^{th} block. The latter task can be done efficiently via polynomial interpolation.

3.2.2 Proof of Robustness

The proof of robustness is similar to [AIKW15] with two modifications. First, we prove robustness (while in [AIKW15] they only proved security for one key); however this does not change the proof.

[6]In any case, RSA security breaks for such output lengths (or even smaller ones). When m is larger, we can assume that S sets all the e_i's to 1, and the pPRG generates the string $(\mathsf{hc}(u, r), \dots, \mathsf{hc}(u, r))$.

Second, we show that the proof remains valid even when the primes $e_1, \ldots, e_m$ are sampled via an admissible sampler whose seed is given to the adversary.

We start by quoting the RSA assumption parameterized with a time-bound $t = t(\lambda)$.

Assumption 3.7 (The RSA assumption). *For every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$ (that tries to break RSA), and all sufficiently large λ,*

$$\Pr\left[\begin{array}{ll} p, q \leftarrow_U \text{primes of length } \lambda; N \leftarrow p \cdot q; & \\ e \leftarrow_U \mathbb{Z}^*_{\varphi(N)}; y \leftarrow_U \mathbb{Z}^*_N; & : \quad \mathcal{A}((N,e),z) = y \\ z \leftarrow y^e \bmod N; & \end{array}\right] \leq \frac{1}{t(\lambda)}.$$

The polynomial variant of the assumption refers to the case where the assumption holds for every polynomial $t(\lambda)$ (in this case, the probability is upper-bounded by a negligible function), and the sub-exponential version refers to the case where $t = 2^{\Omega(\lambda^\delta)}$ for some constant $\delta > 0$.

Recall that in the construction of pPRG from the RSA assumption we only use prime exponents e. Since the density of primes in $\mathbb{Z}^*_{\varphi(N)}$ is $\Theta(1/\log N)$, the RSA assumption implies the following assumption.

Assumption 3.8 (The RSA assumption for prime exponents). *For every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$ (that tries to break RSA)*

$$\Pr\left[\begin{array}{ll} p, q \leftarrow_U \text{primes of length } \lambda; N \leftarrow p \cdot q; & \\ e \leftarrow_U \text{primes of length } \lambda; y \leftarrow_U \mathbb{Z}^*_N; & : \quad \mathcal{A}((N,e),z) = y \\ z \leftarrow y^e \bmod N; & \end{array}\right] \leq \frac{1}{t(\lambda)}.$$

By the security of the Goldreich-Levin hardcore bit [GL89], it is hard to compute the bit $\mathsf{hc}(y, r)$ given $N, e, r, z \leftarrow y^e \bmod N$.

Proposition 3.9 (The GL hardcore for RSA [GL89]). *Let $\mathcal{C}$ be an algorithm whose inputs are N, e, r, z (this algorithm tries to compute the GL hardcore bit), where $N, r \in \{0,1\}^\lambda$ for some $\lambda \in \mathbb{N}$, $e \in \mathbb{Z}^*_{\varphi(N)}$, and $z \in \mathbb{Z}^*_N$ and let*

$$\mathrm{GOOD} = \left\{(N,e,y) : \Pr_r[\mathcal{C}(N,e,r,y^e \bmod N) = \mathsf{hc}(r,y)] \geq 1/2 + \varepsilon\right\}$$

for some ε. There is a non-uniform algorithm $\mathcal{D}(N, e, z = y^e \bmod N)$ that uses $\mathrm{poly}(\lambda, 1/\varepsilon)$ calls to $\mathcal{C}$, each call is with N, e, z, and a uniformly distributed r, such that

$$\Pr[\mathcal{D}(N,e,y^e \bmod N) = y] > 3/4$$

for every $N, e, y \in \mathrm{GOOD}$ (i.e., $\mathcal{D}$ breaks the RSA for such inputs).

Lemma 3.10. *Under the sub-exponential (resp., polynomial) RSA assumption, the above pPRG is sub-exponential-robust pPRG (resp., polynomial-robust pPRG) with sub-exponential stretch (resp., arbitrary polynomial stretch).*

Proof. Fix λ and let $\mathcal{B}$ be any non-uniform sub-exponential time (resp. poly-time) adversary that breaks the robustness of the pPRG based on the RSA assumption with advantage $\varepsilon = \varepsilon(\lambda)$. We will construct a non-uniform adversary $\mathcal{A}$ that breaks the RSA assumption (for prime exponents) with probability ε/m and runs in time polynomial in $\lambda, 1/\varepsilon$, and in the running time of $\mathcal{B}$. Let m be a length parameter and $T_1, \ldots, T_\ell$ be the sets such that $\mathcal{B}(1^\lambda)$ sends $m, T_1, \ldots, T_\ell$ to the challenger in the first step. Define $T^* = [m] \setminus \cup_{1 \le i \le t} T_i$. To simplify notation assume that $T^* = \{1, \ldots, \tau\}$ for some $0 \le \tau < m$. Below we describe an algorithm $\mathcal{A}(N, e, z \leftarrow y^e \bmod N)$ that tries to break the RSA assumption.

To describe $\mathcal{A}$, we first define $\tau + 1$ hybrids $H_0, \ldots, H_\tau$. The ℓ^{th} hybrid is the following distribution:

- Let $\mathsf{params} = (N, r, \rho, u)$ be generated as in the setup algorithm of the pPRG, that is, N is a product of two uniformly distributed primes p, q of length λ and r, ρ, u are uniformly distributed in the appropriate sets, and let $(e_1, \ldots, e_m) \leftarrow S(\rho)$.
- The values $a\{T_1\}, \ldots, a\{T_\ell\}$ and $c_1, \ldots, c_\ell$ are computed as in the pPRG with seed N, r, ρ, u, that is, $a\{T_i\} \leftarrow u^{\prod_{j \in T_i} 1/e_j} \bmod N$ and $c_i \leftarrow \mathsf{hc}(u^{1/e_i} \bmod N, r)$.
- Sample $c_{\ell+1}, \ldots, c_\tau$, where $c_i \in_U \{0,1\}$ (i.e., c_i is a uniformly distributed random bit).
- Output $\mathsf{params} = (N, u, r, \rho), a\{T_1\}, \ldots, a\{T_\ell\}, c_1, \ldots, c_\tau$.

Let

$$p_\ell = \Pr_{N, r, \rho, c_1, \ldots, c_\tau, a\{T_1\}, \ldots, a\{T_\ell\} \leftarrow H_\ell} [\mathcal{B}(N, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, c_1, \ldots, c_\tau) = 1].$$

In other words, in H_ℓ the first ℓ bits are pseudorandom while the last $\tau - \ell$ bits are random. Observe that H_τ corresponds to the message viewed by the adversary $\mathcal{B}$ in the robust-pPRG game when the challenger's bit is $b = 1$ and H_0 corresponds to the message viewed by the adversary $\mathcal{B}$ when the challenger's bit is $b = 0$. That is, the adversary wins the robustness game of the pPRG if when getting a sample from H_τ it answers 1 or when it gets a sample from H_0 it answers 0. As $\mathcal{B}$ wins the robustness game of the pPRG with probability at least $1/2 + \varepsilon$:

$$0.5 p_\tau + 0.5(1 - p_0) \ge 0.5 + \varepsilon, \tag{1}$$

That is, $p_\tau - p_0 \ge 2\varepsilon$.

Thus, there exists an index $\ell \in \{1, \ldots, \tau\}$ such that $\mathcal{B}$ distinguishes between samples from H_ℓ and samples from $H_{\ell-1}$ with probability at least $\varepsilon/\tau > \varepsilon/m$, that is, without loss of generality

$$\left(\begin{array}{l} \Pr_{H_\ell}[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, c_1, \ldots, c_\tau) = 1] \\ \quad - \quad \Pr_{H_{\ell-1}}[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, c_1, \ldots, c_\tau) = 1] \end{array} \right) \ge 2\varepsilon/m. \tag{2}$$

Define GOOD_0 as the set of all (N, u, ρ) such that the difference of the probabilities in (2) conditioned on N, u, ρ is at least ε/m (the first probability is over the choice of r and $c_{\ell+1}, \ldots, c_\tau$ and the second probability is also over the choice of c_ℓ). Thus,

$$\Pr[(N, u, \rho) \in \mathrm{GOOD}_0] \ge \varepsilon/m.$$

Construct $\mathcal{A}(N, e, z \leftarrow y^e \bmod N)$ that tries to invert the RSA encryption as follows:

1. Using the planting algorithm, generate a random ρ such that $(e_1, \ldots, e_m) \leftarrow S(\rho)$ and $e_\ell = e$.
2. Let $u \leftarrow z^{\prod_{i \neq \ell} e_i} \bmod N$, $a\{T_j\} \leftarrow z^{\prod_{i \notin T_j, i \neq \ell} e_i} \bmod N$ for $1 \leq j \leq \ell$, and $y_j \leftarrow z^{\prod_{i \neq \ell, j} e_i} \bmod N$ for $1 \leq j \leq m, j \neq \ell$.

   (* $a\{T_j\} \equiv u^{\prod_{i \in T_j} 1/e_i} \pmod N$ and $y_j \equiv u^{1/e_j} \pmod N$ as required. *)
3. Denote $\tilde{y} = y^{\prod_{i \neq \ell} e_i} \bmod N$ (* $u \equiv \tilde{y}^e \pmod N$; this notation is only used in the proof ($\mathcal{A}$ does not know y) *).
4. Find $\tilde{y}$ using the GL algorithm (Proposition 3.9), where every hardcore query $r, u \equiv \tilde{y}^e \bmod N$ is answered as follows:
   (a) Let $\sigma \leftarrow_U \{0,1\}$ and compute for every $1 \leq i \leq m$
   $$C_i = \begin{cases} \mathsf{hc}(r, y_i) & \text{if } 1 \leq i < \ell \\ \sigma & \text{if } i = \ell \\ r_i \leftarrow_U \{0,1\} & \text{if } \ell < i \leq \tau \end{cases}$$
   (b) If $\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_\tau) = 1$ answer σ, else answer $\overline{\sigma}$.
5. Let $\hat{y}$ be the output of the GL algorithm. If $u \not\equiv \hat{y}^e \pmod N$, output "FAIL" and halt.
6. Otherwise, compute y from $\hat{y} \equiv y^{\prod_{i \neq \ell} e_i} \pmod N$ and $z \equiv y^e \pmod N$ using Shamir's algorithm [Sha81], that is, find the integers coefficients a, b of Bézout's identity $a \prod_{i \neq \ell} e_i + be = \gcd(\prod_{i \neq \ell} e_i, e) = 1$ using the extended Euclidean algorithm and compute $y \leftarrow \hat{y}^a \cdot z^b \bmod N$.

In $\mathcal{A}$, the elements N, e are chosen with uniform distribution and ρ is a random element conditioned on $e_\ell = e$, thus, N, ρ are random elements. Furthermore, y is a random element and $y^{\prod_{1 \leq i \leq m} e_i}$ is a permutation, thus, the element u computed in $\mathcal{A}$ is a random element in $\mathbb{Z}_N^*$. We analyze the probability that $\mathcal{A}$ on a query r, u in step (4b) outputs $\mathsf{hc}(y, u)$.

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ outputs } \mathsf{hc}(r,u)] \\
&\quad = 1/2 \cdot \Pr[\mathcal{A} \text{ outputs } \mathsf{hc}(u,r) | \sigma = \mathsf{hc}(r,u)] + 1/2 \cdot \Pr[\mathcal{A} \text{ outputs } \mathsf{hc}(u,r) | \sigma = \overline{\mathsf{hc}(r,u)}] \\
&\quad = 1/2 \cdot \Pr[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_{\ell-1}, \mathsf{hc}(u,r), C_{\ell+1}, \ldots, C_\tau) = 1] \\
&\qquad + 1/2 \cdot \Pr[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_{\ell-1}, \overline{\mathsf{hc}(u,r)}, C_{\ell+1}, \ldots, C_\tau) = 0] \\
&\quad = \Pr[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_{\ell-1}, \mathsf{hc}(u,r), C_{\ell+1}, \ldots, C_\tau) = 1] \\
&\qquad - 1/2 \Pr[\mathcal{B}(N, u, r, \tau, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_{\ell-1}, \mathsf{hc}(u,r), C_{\ell+1}, \ldots, C_\tau) = 1] \\
&\qquad + 1/2 - 1/2 \cdot \Pr[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_{\ell-1}, \overline{\mathsf{hc}(u,r)}, C_{\ell+1}, \ldots, C_\tau) = 1].
\end{aligned}$$

Note that

$$\Pr[\mathcal{B}(N, u, r, \rho, a\{T_1\}, \ldots, a\{T_\ell\}, C_1, \ldots, C_{\ell-1}, \mathsf{hc}(u,r), C_{\ell+1}, \ldots, C_\tau) = 1]$$

is the probability that $\mathcal{B}$ outputs 1 on a sample from H_ℓ while

$$1/2\Pr[\mathcal{B}(N,u,r,\rho,a\{T_1\},\dots,a\{T_\ell\},C_1,\dots,C_{\ell-1},\mathsf{hc}(u,r),C_{\ell+1},\dots,C_\tau)=1] \\ +1/2\cdot\Pr[\mathcal{B}(N,u,r,\rho,a\{T_1\},\dots,a\{T_\ell\},C_1,\dots,C_{\ell-1},\overline{\mathsf{hc}(u,r)},C_{\ell+1},\dots,C_\tau)=1]$$

is the probability that $\mathcal{B}$ outputs 1 on a sample from $H_{\ell-1}$. Thus, conditioned on the event that $(N,u,\rho)\in \mathrm{GOOD}_0$, an event with probability at least ε/m, the probability that $\mathcal{A}$ answers $\mathsf{hc}(r,u)$ is at least ε/m. By Proposition 3.9, for $(N,u,\rho)\in\mathrm{GOOD}_0$ algorithm $\mathcal{A}$ finds $\hat{y}\equiv u^{1/e}\equiv y^{\prod i\neq\ell e_i}$ $(\mathrm{mod}\ N)$ with probability at least $3/4$. By the correctness of Shamir's algorithm, in this case algorithm $\mathcal{A}$ outputs the required y. I.e., if $\mathcal{B}$ breaks the pPRG with advantage ε, then $\mathcal{A}$ breaks the RSA assumption with probability at least $\frac{3\varepsilon}{4m}$.

To conclude, given any (sub-exponential) 2^{λ^δ}-time adversary $\mathcal{B}$ that breaks the robustness of the pPRG with advantage $1/2^{\lambda^\delta}$, we constructed a sub-exponential adversary $\mathcal{A}$ that breaks the RSA assumption with probability $3/(4\cdot 2^{\lambda^\delta}\cdot m)$. Since $\mathcal{B}$ is sub-exponential, the length $m=m(\lambda)$ is also sub exponential. By the sub-exponential $2^{2\lambda^\delta}$-time RSA assumption such $\mathcal{A}$ cannot exists and robustness follows. □

3.3 Projective PRG from iO and SSB Hash Functions

We now show a construction of projective PRGs from indistinguishability obfuscation (iO) [BGI+01, GGH+13] and somewhere statistically binding (SSB) hash functions [HW15]. We will in fact show how to take any puncturable pseudorandom function family and construct from it a projective PRG using iO and SSB hash functions. Our ingredients are the following.

- An indistinguishability obfuscator $\mathcal{O}$ [BGI+01, GGH+13];
- An SSB hash function family $\mathcal{SSB}:=(\mathsf{SSB.Gen},\mathsf{SSB}.H,\mathsf{SSB.Open},\mathsf{SSB.Ver})$ [HW15]; and
- A puncturable PRF family $\mathcal{F}_n:=\{F_k : k\in\{0,1\}^\lambda\}$ [BGI14, KPTZ13, BW13] which can be constructed from OWF and therefore any SSB hash function family.

Given these ingredients, our construction proceeds as follows.

- The seed for the projective PRG is a uniformly random seed for the puncturable PRF $k\leftarrow\{0,1\}^\lambda$. The output of the PRG is

  $$G(k)=(F_k(0)||F_k(1)||\dots||F_k(m))\ .$$

- The projective key $k\{T\}$ is computed as follows. Consider the program $\Pi_{h,y,k}$ which has in it an SSB hash function $h\leftarrow\mathsf{SSB.Gen}(1^\lambda)$, the hash $y\leftarrow\mathsf{SSB}.H(h,T)$ of the characteristic vector of the set T using h, and the seed k.

  The program takes as input an index i together with an SSB opening ρ_i to the value $T[i]=1$, and checks it against the root y. That is, it checks whether $\mathsf{SSB.Ver}(h,y,i,\rho_i)=1$. If the check passes, the program computes $F_k(i)$ and outputs it, otherwise it outputs $\perp$.

  The projective key $k\{T\}$ is the obfuscated program $\mathcal{O}(\Pi_{h,y,k})$.

- Running the program $k\{T\}$ on input $i \in T$ together with the SSB opening of leaf i recovers the i-th output bit of the projective PRG, simply by the functionality of the program $\Pi_{h,y,k}$.

Theorem 3.11. *The above construction is a projective pseudorandom generator assuming that $\mathcal{F}_n$ is a projective PRF family, $\mathcal{O}$ is an indistinguishability obfuscator and $\mathcal{SSB}$ is an SSB hash function family. The projective keys have size* $\mathsf{poly}(\log m, \lambda)$.

Proof. (of Theorem 3.11.) The proof goes by a hybrid argument that is by now standard in the iO literature. Assume that the adversary is given projective keys $k\{T_1\}, \ldots, k\{T_\ell\}$ and either $G(k)_{\overline{T}}$ or a random string of the same length, where $T = T_1 \cup T_2 \cup \ldots \cup T_\ell$. We wish to show that no ppt adversary can distinguish between the two possibilities. We show this using $m - |T| + 1$ hybrids by iterating over indices $j \in \overline{T}$.

Hybrid j, for $j \in \{0, 1, \ldots, |\overline{T}|\}$. The adversary is given projective keys $k\{T_1\}, \ldots, k\{T_\ell\}$ and a string $z \in \{0,1\}^{|\overline{T}|}$ whose first j bits are uniformly random and whose last $m - j$ bits are the last $m - j$ bits of $G(k)_{\overline{T}}$.

Note that hybrid $|\overline{T}|$ corresponds to the adversary receiving a uniformly random string z and hybrid 0 to the adversary receiving $z = G(k)_{\overline{T}}$.

Claim 3.12. *For $j \in \{0, 1, \ldots, |\overline{T}|-1\}$, hybrids j and $j+1$ are computationally indistinguishable under the assumptions of Theorem 3.11.*

Proof. We construct the following hybrids.

Hybrid $j.0$. This is the same as Hybrid j.

Hybrid $j.1$. Change the hash function to be binding on index $j + 1$. That is, let

$$h \leftarrow \mathsf{SSB.Gen}(1^\lambda, j + 1)$$

Hybrids $j.0$ and $j.1$ are computationally indistinguishable due to the indistinguishability of SSB keys.

Hybrid $j.2$. Puncture the PRF key k on input $j + 1$, and let the program $\Pi_{h,y,k\{j+1\},\beta=F_k(j+1)}$ be defined exactly as $\Pi_{h,y,k}$ except on input $j + 1$, it outputs β.

Hybrids $j.1$ and $j.2$ are computationally indistinguishable by the security of indistinguishability obfuscation and the fact that the two programs in question are functionally identical.

Hybrid $j.3$. Change β to $\perp$.

Hybrids $j.2$ and $j.3$ are computationally indistinguishable by the security of indistinguishability obfuscation and the fact that there are *no* SSB openings to location $j + 1$ to the value $T[j + 1] = 1$. Consequently, both programs in question output $\perp$ on input $j + 1$.

Hybrid $j.4$. Change the $(j + 1)$-th bit of the string z to a uniformly random bit.

Hybrids $j.3$ and $j.4$ are computationally indistinguishable by the security of the puncturable PRF.

Hybrid $j.5$. Replace the punctured key with the true PRF key.

Hybrids $j.4$ and $j.5$ are computationally indistinguishable by the security of indistinguishability obfuscation scheme.

Hybrid $j.6$. Replace the hash key with one that is generated as $h \leftarrow \mathsf{SSB.Gen}(1^\lambda)$.

Hybrids $j.5$ and $j.6$ are computationally indistinguishable due to the indistinguishability of SSB hash keys.

Finally, hybrid $j.6$ is the same as hybrid $j+1$, an observation that finishes the proof of the claim. □

Since every adjacent pair of hybrids is computationally indistinguishable, an adversary can win the projective PRG game with advantage only $O(m)$ more than the advantage in any of the security games: for the SSH hash, the projective PRG or the iO scheme. This finishes the proof of the theorem. □

3.4 Projective PRG from the Diffie-Hellman Assumption

We first describe a projective PRG based on the decisional Diffie-Hellman (DDH) assumption. The construction results in small projective keys, but large (non-reusable) public parameters of size quadratic in the output length of the PRG. Later in this section, we show a different construction using bilinear maps that has reusable public parameters. This latter construction also admits a "balancing trick" that allows us to trade off the size of the public parameters for the size of the projective key.

3.4.1 A Construction from DDH-Hard Groups

We start with the DDH assumption.

Assumption 3.13 (The decisional Diffie-Hellman assumption). *Let $\mathcal{G} = \{\mathbb{G}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of (Abelian) groups of prime order $q = q(\lambda)$. The (t,ε)-decisional Diffie-Hellman (DDH) assumption with respect to $\mathcal{G}$ requires that for every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$,*

$$\Pr_{a,b\leftarrow[q]}\left[\mathcal{A}(g,g^a,g^b,g^{ab})=1\right] - \Pr_{a,b,c\leftarrow[q]}\left[\mathcal{A}(g,g^a,g^b,g^c)=1\right] \le \varepsilon(\lambda),$$

where g is any generator of $\mathbb{G}_\lambda$. If the (t,ε)-DDH assumption holds with respect to $\mathcal{G}$, we refer to $\mathcal{G}$ as a DDH-hard (family of) groups.

Theorem 3.14. *Under the DDH assumption, there exists a robust pPRG whose projective keys are strongly succinct, namely, of length $O(\lambda)$. The public parameter has length $m^2 \cdot \mathsf{poly}(\lambda)$, where m is the output length of the pPRG.*

Proof. Our pPRG construction is inspired by [AIKW15].

Setup. For a security parameter λ, let $\mathbb{G}_\lambda$ be a DDH-hard group. For an output length m, we sample

$$\mathsf{msk} \leftarrow (x_1,\ldots,x_m,y_1,\ldots,y_m) \in \mathbb{Z}_q^{2m},$$

Sample elements $r_1, \ldots, r_m \in \mathbb{Z}_q$ and set params $\leftarrow (q, g, \mathbf{v}, \mathbf{M})$ as

$$q, g, \mathbf{v} = \begin{bmatrix} g^{r_1} \\ g^{r_2} \\ \vdots \\ g^{r_m} \end{bmatrix} \text{ and } \mathbf{M} = \begin{bmatrix} g^{r_1 x_1} g^{y_1} & g^{r_1 x_2} & \cdots & g^{r_1 x_m} \\ g^{r_2 x_1} & g^{r_2 x_2} g^{y_2} & \cdots & g^{r_2 x_m} \\ \vdots & \vdots & \ddots & \vdots \\ g^{r_m x_1} & g^{r_m x_2} & \cdots & g^{r_m x_m} g^{y_m} \end{bmatrix}.$$

The i^{th} output of the projective PRG is g^{y_i}. We note that the master secret key msk is long as defined, but this is compatible with the pPRG definition that does not put any requirement on the length of msk. (Moreover, one can always use an auxiliary ordinary PRG to generate msk and set the seed of that PRG as the master secret key.)

Key Generation. Given the master key msk $= (x_1, \ldots, x_m, y_1, \ldots, y_m)$ and a set $T \subseteq [m]$, the projective key is $a\{T\} = \sum_{i \in T} x_i$.

Evaluation. Given params $= (q, g, \mathbf{v}, \mathbf{M})$, a set $T \subseteq [m]$, and a projective key $a\{T\}$, we recover the output g^{y_i} for $i \in T$ by computing

$$\left(\prod_{j \in T} \mathbf{M}_{i,j} \right) \cdot (v_i)^{-a\{T\}}.$$

The correctness of the construction follows from the following calculation:

$$\left(\prod_{j \in T} \mathbf{M}_{i,j} \right) \cdot (v_i)^{-a\{T\}} = \left(\prod_{j \in T} g^{r_i x_j} \right) \cdot g^{y_i} \cdot (g^{r_i})^{-\sum_{j \in T} x_j} = (g^{r_i})^{\sum_{j \in T} x_j} \cdot g^{y_i} \cdot (g^{r_i})^{-\sum_{j \in T} x_j} = g^{y_i},$$

where the first equation holds as long as $i \in T$.

For robustness, assume that the adversary has received the projective keys for subsets $T_1, T_2, \ldots, T_\ell$, and let $T = \bigcup_j T_j$. All the revealed projective keys can be computed from $(x_i)_{i \in T}$. Without loss of generality, assume that $T = \{k+1, \ldots, m\}$. It suffices to show that $(g^{y_i})_{i \notin T}$ is computationally hidden from the adversary, who gets $(x_i)_{i \in T}$, as well as $p, g, \mathbf{v}$, and $\mathbf{M}_{1:m,1:k}$, where the latter denotes the first k columns of $\mathbf{M}$. (Note that the last $m-k$ columns of $\mathbf{M}$ can be computed from $\mathbf{v}$ and $x_{k+1}, \ldots, x_m$).

The proof relies on the DDH assumption (Assumption 3.13), which implies the following matrix-DDH assumption [BHHO08]:

$$q, g, \begin{bmatrix} g^{r_1} \\ \vdots \\ g^{r_m} \end{bmatrix}, \begin{bmatrix} g^{x_1} \\ \vdots \\ g^{x_k} \end{bmatrix}, \begin{bmatrix} g^{r_1 x_1} & \cdots & g^{r_1 x_k} \\ \vdots & \ddots & \vdots \\ g^{r_m x_1} & \cdots & g^{r_m x_k} \end{bmatrix} \approx_{\mathsf{c}} q, g, \begin{bmatrix} g^{r_1} \\ \vdots \\ g^{r_m} \end{bmatrix}, \begin{bmatrix} g^{x_1} \\ \vdots \\ g^{x_k} \end{bmatrix}, \begin{bmatrix} g^{u_{1,1}} & \cdots & g^{u_{1,k}} \\ \vdots & \ddots & \vdots \\ g^{u_{m,1}} & \cdots & g^{u_{m,k}} \end{bmatrix}$$

where $x_1, \ldots, x_k, r_1, \ldots, r_m, u_{1,1}, \ldots, u_{m,k}$ are independently randomly sampled. Thus

$$(q, g, \mathbf{v}, \mathbf{M}_{1:m,1:k}) \approx_{\mathsf{c}} (q, g, \mathbf{v}, \mathbf{U} \odot \mathbf{Y})$$

where $\mathbf{U} \in \mathbb{G}^{m \times k}$ consists of uniformly random group elements, $\odot$ stands for an entry-wise product, and $\mathbf{Y}$ is a diagonal m-by-k matrix whose $(i,i)^{th}$ entry, for $i \leq k$, is g^{y_i} and the rest of the entries are all 1. The latter distribution statistically hides $(y_i)_{i \notin T}$. $\square$

We can apply a balancing trick to achieve a trade-off between the projective-key's length and the public parameter's length: Start from a robust pPRG whose projective keys and public parameters have length $\ell_{\mathsf{key}}(m)$ and $\ell_{\mathsf{param}}(m)$, respectively, and fix $\delta \in [0,1]$. Then for a shorter output length $m' = m^{1-\delta}$, the public parameter (resp. projective key) has length $\ell_{\mathsf{param}}(m^{1-\delta})$ (resp. $\ell_{\mathsf{key}}(m^{1-\delta})$). The concatenation of m^{δ} such shorter pPRG is a robust pPRG of output length m. The resulting robust pPRG has projective key length $m^{\delta} \cdot \ell_{\mathsf{key}}(m^{1-\delta})$ and public parameter length $m^{\delta} \cdot \ell_{\mathsf{param}}(m^{1-\delta})$. We obtain the following corollary.

Corollary 3.15. *Under the DDH assumption, for every $\delta \in [0,1]$ there exists a robust pPRG whose projective keys have length $m^{\delta} \cdot \mathrm{poly}(\lambda)$ and the public parameter has length $m^{2-\delta} \cdot \mathrm{poly}(\lambda)$, where m is the output length. Furthermore, the evaluation algorithm needs only $m^{2-2\delta} \cdot \mathrm{poly}(\lambda)$ bits from the public parameter to evaluate a single bit c_i.*

3.4.2 Reusable Public Parameters via Bilinear Maps

We describe a projective PRG based on the bilinear DDH assumption. The improvement over the DDH-based construction is that the (long) public parameters are independent of the PRG seed and are reusable.

Assumption 3.16 (The decisional bilinear Diffie-Hellman assumption). *Let $\mathcal{PG} = \{(\mathbb{G}_\lambda, \mathbb{G}_{T,\lambda}, q_\lambda, e_\lambda)\}_{\lambda \in \mathbb{N}}$ be a family of pairing groups of prime order $q = q(\lambda)$. The (t,ε)-decisional bilinear Diffie-Hellman (DBDH) assumption with respect to $\mathcal{PG}$ requires that for every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$,*

$$\Pr_{a,b,c \leftarrow [q]}\left[\mathcal{A}(g, g^a, g^b, g^c, e(g,g)^{abc}) = 1\right] - \Pr_{a,b,c,d \leftarrow [q]}\left[\mathcal{A}(g, g^a, g^b, g^c, e(g,g)^d = 1\right] \leq \varepsilon(\lambda),$$

where g is any generator of $\mathbb{G}_\lambda$. If (t,ε)-DBDH assumption holds with respect to $\mathcal{PG}$, we refer to $\mathcal{PG}$ as a DBDH-hard (family of) pairing groups.

Theorem 3.17. *Under the DBDH assumption, there exists a robust pPRG whose projective keys are strongly succinct, i.e., of length $O(\lambda)$. The public parameter is of length $m^2 \cdot \mathsf{poly}(\lambda)$, where m is the output length. The public parameter is independent of the PRG master secret key* msk *and is reusable.*

Proof. The construction of Theorem 3.17 is as follows.

- For the given security parameter, let $\mathbb{G}, \mathbb{G}_{\mathrm{T}}$ be the corresponding pairing groups of order $q = q(\lambda)$, let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}$ be bilinear map.
- The parameter sampler algorithm chooses a generator $g \in \mathbb{G}$, samples random $r_1, \ldots, r_m$, $x_1, \ldots, x_m, y_1, \ldots, y_m \leftarrow [q]$ and creates the AIKW matrix $\mathbf{M} \in \mathbb{G}^{m \times m}$ whose $(i,j)^{\text{th}}$ entry is $g^{r_i x_j}$ if $i \neq j$ and $g^{r_i x_j} \cdot g^{y_i}$ if $i = j$. The public parameters consist of g, $\mathbf{M}$, g^{r_i} and g^{x_j} for all $i, j \in [m]$. Crucially, the y_i's are not revealed.
- The msk-sampler algorithm samples an element $s \in_U \mathbb{Z}_q$ and sets msk $\leftarrow s$.

  Note that the public parameters do not depend on the master secret key msk. Indeed, we will show that they are reusable across many different PRG seeds.

- The projective key for a master secret key s and a subset $T \subseteq [n]$ is

$$a\{T\} = \left(g^s, g^{s \cdot (\sum_{j \in T} x_j)}\right).$$

- We define the i^{th} output of the PRG to be $e(g,g)^{y_i s}$. Given a projective key $a\{T\} = (g^s, g^{s \cdot (\sum_{j \in T} x_j)})$, we can recover the i^{th} output, if $i \in T$, by computing

$$e\Big(g^s, \prod_{j \in T} \mathbf{M}_{i,j}\Big) \Big/ e\Big(\underbrace{g^{s \cdot \sum_{j \in T} x_j}}_{\text{from } a\{T\}}, g^{r_i}\Big).$$

The correctness is easy to verify. Note that $i \in T$ implies $\prod_{j \in T} \mathbf{M}_{i,j} = g^{r_i \cdot (\sum_{i \in T} x_i)} \cdot g^{y_i}$. Thus

$$\begin{aligned} \frac{e\Big(g^s, \prod_{j \in T} \mathbf{M}_{i,j}\Big)}{e\Big(g^{s \cdot \sum_{j \in T} x_j}, g^{r_i}\Big)} &= \frac{e\Big(g^s, g^{r_i \cdot (\sum_{i \in T} x_i)} \cdot g^{y_i}\Big)}{e\Big(g^{s \cdot \sum_{j \in T} x_j}, g^{r_i}\Big)} \\ &= \frac{e(g,g)^{s r_i \cdot \sum_{j \in T} x_j + s y_i}}{e(g,g)^{s r_i \cdot \sum_{j \in T} x_j}} \\ &= e(g,g)^{s y_i}, \end{aligned}$$

which is the i^{th} output for the master secret key s.

We now show robustness, starting from the case when the public parameter is not reused. Assume that the adversary has received the projective keys for $T_1, T_2, \ldots, T_\ell$ and let $T = \bigcup_j T_j$. All the revealed projective keys can be computed from $g^s, (g^{s x_i})_{i \in T}$. It suffices to show that given $g^s, (g^{s x_i})_{i \in T}$ and the public parameter, the information of $(e(g,g)^{s y_i})_{i \notin T}$ is still computationally hidden. Concretely, it suffices to show that when all r_i, x_i, u_i are i.i.d. random,

$$\begin{aligned} &g, (g^{r_i})_{i \in [m]}, (g^{x_i})_{i \in [m]}, g^s, (g^{s x_j})_{j \in T}, \mathbf{M}, (e(g,g)^{s y_j})_{j \notin T} \\ \approx_{\mathsf{c}}\ &g, (g^{r_i})_{i \in [m]}, (g^{x_i})_{i \in [m]}, g^s, (g^{s x_j})_{j \in T}, \mathbf{M}, (e(g,g)^{u_j})_{j \notin T}. \end{aligned} \tag{3}$$

By the standard hybrid argument, it suffices to show that for all $t \notin T$,

$$\begin{aligned} &g, (g^{r_i})_{i \in [m]}, (g^{x_i})_{i \in [m]}, g^s, (g^{s x_j})_{j \in T}, \mathbf{M}, (e(g,g)^{s y_j \text{ if } j \le t;\ u_j \text{ if } j > t})_{j \notin T} \\ \approx_{\mathsf{c}}\ &g, (g^{r_i})_{i \in [m]}, (g^{x_i})_{i \in [m]}, g^s, (g^{s x_j})_{j \in T}, \mathbf{M}, (e(g,g)^{s y_j \text{ if } j < t;\ u_j \text{ if } j \ge t})_{j \notin T}. \end{aligned}$$

We now show that the above two distributions are computationally indistinguishable even if the adversary additionally gets $(r_j)_{j \ne t}, (x_j)_{j \ne t}, (y_j)_{j \ne t}$. Note that, the adversary can perfectly simulate part of its view, by randomly sampling $(r_j)_{j \ne t}, (x_j)_{j \ne t}, (y_j)_{j \ne t}$ and computing by its own $g^{r_j}, g^{x_j}, g^{s x_j}$ for all $j \ne t$ and the matrix $\mathbf{M}$ except for $\mathbf{M}_{t,t}$ $(= g^{r_t x_t + y_t})$. So it suffices to show that

$$g, g^{r_t}, g^{x_t}, g^s, g^{r_t x_t + y_t}, e(g,g)^{s y_t} \approx_{\mathsf{c}} g, g^{r_t}, g^{x_t}, g^s, g^{r_t x_t + y_t}, e(g,g)^u.$$

By the DBDH assumption, given $g, g^{r_t}, g^{x_t}, g^s, g^{r_t x_t + y_t}$ (the last one is one-time padded by y_t), the distribution of $e(g,g)^{s r_t x_t}$ is indistinguishable from uniform distribution over $\mathbb{G}_{\mathrm{T}}$. Then

$$e(g,g)^{s y_t} = e(g^s, g^{r_t x_t + y_t}) \Big/ e(g,g)^{s r_t x_t}$$

is also indistinguishable from a random group element, given $g, g^{r_t}, g^{x_t}, g^s, g^{r_t x_t + y_t}$, since it is one-time padded by $e(g,g)^{s r_t x_t}$.

For the reusable robustness, considering randomly sampled seeds $s_1, \ldots, s_k$, assume that for each $i \in [k]$, the adversary has received s_i's projective keys for $T_{i,1}, T_{i,2}, \ldots$ such that $T_i = \bigcup_j T_{i,j}$. All the revealed projective keys can be computed from $(g^{s_i})_{i\in[k]}$, $(g^{s_i x_j})_{i\in[k], j \in T_i}$. It suffices to show that given $g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, (g^{s_i})_{i\in[k]}, (g^{s_i x_j})_{i\in[k], j\in T_i}, \mathbf{M}$, the distribution of $(e(g,g)^{s_i y_j})_{i\in[k], j\notin T_i}$ is computationally indistinguishable from uniform. Concretely, it suffices to show

$$\begin{aligned} & g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, (g^{s_i})_{i\in[k]}, (g^{s_i x_j})_{i\in[k], j\in T_i}, \mathbf{M}, (e(g,g)^{s_i y_j})_{i\in[k], j\notin T_i} \\ \approx_{\mathsf{c}}\ & g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, (g^{s_i})_{i\in[k]}, (g^{s_i x_j})_{i\in[k], j\in T_i}, \mathbf{M}, (e(g,g)^{u_{i,j}})_{i\in[k], j\notin T_i} \,. \end{aligned} \tag{4}$$

We prove (4) using the standard hybrid argument, by showing that for any $t \in [k]$,

$$\begin{aligned} & g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, (g^{s_i})_{i\in[k]}, (g^{s_i x_j})_{i\in[k], j\in T_i}, \mathbf{M}, (e(g,g)^{s_i y_j \text{ if } i \le t;\ u_{i,j} \text{ if } i > t})_{i\in[k], j\notin T_i} \\ \approx_{\mathsf{c}}\ & g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, (g^{s_i})_{i\in[k]}, (g^{s_i x_j})_{i\in[k], j\in T_i}, \mathbf{M}, (e(g,g)^{s_i y_j \text{ if } i < t;\ u_{i,j} \text{ if } i \ge t})_{i\in[k], j\notin T_i} \,. \end{aligned}$$

The two distributions are indistinguishable even if the adversary additionally gets $(s_i)_{i\neq t}$. Note that, since the adversary gets $(s_i)_{i\neq t}$, the adversary is essentially distinguishing the following two distributions. (The terms that are perfectly simulatable by the adversary are ignored.)

$$\begin{aligned} & g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, g^{s_t}, (g^{s_t x_j})_{j\in T_t}, \mathbf{M}, (e(g,g)^{s_t y_j})_{j\notin T_t} \\ \approx_{\mathsf{c}}\ & g, (g^{r_i})_{i\in[m]}, (g^{x_i})_{i\in[m]}, g^{s_t}, (g^{s_t x_j})_{j\in T_t}, \mathbf{M}, (e(g,g)^{u_{t,j}})_{j\notin T_t} \,. \end{aligned}$$

This is already proved in the case when the public parameter is not reused, as shown in (3).

□

Since the public parameters are reusable, we can apply a better balancing trick than in Corollary 3.15. Start from a robust pPRG whose projective keys have length $\ell_{\mathsf{key}}(m)$ and whose public parameter is reusable and has length $\ell_{\mathsf{param}}(m)$. Then for a shorter output length $m' = m^{1-\delta}$, the public parameter (resp. projective key) has length $\ell_{\mathsf{param}}(m^{1-\delta})$ (resp. $\ell_{\mathsf{key}}(m^{1-\delta})$). The concatenation of m^δ such shorter pPRG is a robust pPRG of output length m. The resulting robust pPRG has projective key length $m^\delta \cdot \ell_{\mathsf{key}}(m^{1-\delta})$ and public parameter length $\ell_{\mathsf{param}}(m^{1-\delta})$ because the public parameter is reusable. We obtain the following corollary.

Corollary 3.18. *Under the bilinear DDH assumption, for every $\delta \in [0,1]$ there exists a robust pPRG whose projective keys have length $m^\delta \cdot \mathrm{poly}(\lambda)$ and the public parameter has length $m^{2(1-\delta)})\,\mathrm{poly}(\lambda)$, where m is the output length. The public parameter is independent of the PRG master secret key and is reusable.*

4 SCSS from Projective PRGs

In this section, we show how to use pPRGs to construct SCSS for graphs, CNF formulas, and truth tables.

4.1 From Projective PRG to Graph SCSS

Next, we define the notion of secret-sharing schemes for *graphs*. Given an undirected graph $G = (V, E)$, we view the parties as the vertices in V, and the authorized sets are those sets that contain at least one edge from E. That is, using the terminology of Section 2.1, given a graph $G = (V, E)$, where $|V| = \{1, \dots, n\}$, we consider the function $f_G : \{0,1\}^n \to \{0,1\}$ in which $f_G(x) = 1$ if and only if there are $i, j \in [n]$ such that $x_i = x_j = 1$ and $(i, j) \in E$. We will focus on bipartite graph access structures, namely, where V consists of two sets L, R and all edges connect vertices from both sets, i.e. $E \subseteq L \times R$. By [BFM16], efficient secret-sharing schemes for bipartite graphs imply efficient secret-sharing schemes for arbitrary graphs.[7]

We next explain how to construct a secret-sharing scheme for bipartite graphs. We start by describing a secret-sharing scheme for graphs with perfect security, however with share size $O(|V|)$. We will then explain how to compress the shares using a pPRG (settling for computational security). To share a secret $s \in \{0,1\}$ for a bipartite graph $G = (L = \{v_1, \dots, v_m\}, R, E)$, the dealer chooses m random bits $r_1, \dots, r_m$, gives vertex $v_i \in L$ the share $s \oplus r_i$ and gives vertex $w \in R$ the random bits of all its neighbors, that is, denoting $T_w = \{i : (v_i, w) \in E\}$, the share of w is $(r_i)_{i \in T_w}$. Notice that in this scheme a vertex w has to know a subset T_w of the random bits without getting any information on the other bits. This is exactly the functionality supplied by a pPRG.

Theorem 4.1 (Theorem 1.6 restated). *Assume that there is a robust pPRG in which the length of the projective keys is $\kappa(\lambda, m)$ and the length of the public parameters is $\kappa_0(\lambda, m)$, where m is the output length of the generator. Then, for every graph $G = (V, E)$, there is a SCSS with share size $O(\kappa(\lambda, |V|))$ and public information size of $\kappa_0(\lambda, |V|)$.*

Proof. By [BFM16], it suffices to consider bipartite graphs $G = (L, R, E)$. Let $L = \{v_1, \dots, v_m\}$. The sharing algorithm $\mathsf{CSS}.\textsc{Share}(1^\lambda, G, s)$ of the secret-sharing scheme works as follows.

- Compute $(\mathsf{params}, \mathsf{msk}) \leftarrow \text{pPRG}.\textsc{Setup}(1^\lambda, 1^m)$.
- Compute $(c_1, \dots, c_m) \leftarrow \text{pPRG}.\textsc{Eval}(\mathsf{params}, \mathsf{msk})$.
- The public information sh_0 is params.
- The share of $v_i \in L$ is $s \oplus c_i$.
- For a vertex $w \in R$, let $T_w = \{i : (v_i, w) \in E\}$. The share of the vertex w is $a\{T_w\} \leftarrow \text{pPRG}.\textsc{KeyGen}(\mathsf{params}, \mathsf{msk}, T_w)$.

The reconstruction algorithm $\mathsf{CSS}.\textsc{Recon}$ works as follows. Let $x \in \{0,1\}^n$ be an input such that $x_i = x_j = 1$ for $(v_i, w_j) \in E$. We use sh_0 and the share of w_j, namely $a\{T_{w_j}\}$, to decrypt the bit c_i; this is possible since $i \in T_{w_j}$. Computing the exclusive-or of c_i with the share $s \oplus c_i$, which is held by v_i, reconstructs s.

For the security of the scheme, consider an input x and a graph G chosen by the adversary such that S_x does not contain an edge in G. Let $L_x = \{i : v_i \in S_x \cap L\}$ and $R_x = S_x \cap R$. The

[7]Given a graph $G = (V, E)$ construct a bipartite graph H by taking two copies of each vertex and for every $(u, v) \in E$ connect the first copy of u to the second copy of v; the share of each party in V consists of the shares of the corresponding two copies of this vertex in H.

shares of the parties in S_x are $(c_i \oplus s)_{i \in L_x}$ and $(a\{T_w\})_{w \in R_x}$. By the robustness of the pPRG, the bits in $T^* = [m] \setminus \cup_{w \in R_x} T_w$ are indistinguishable in polynomial time from uniformly distributed bits given the projective keys of R_x. Note that for uniformly chosen truly random bits $(r_i)_{i \in T^*}$, the distributions $(r_i)_{i \in T^*}, (a\{T_w\})_{w \in R_x}$ and $(r_i \oplus 1)_{i \in T^*}, (a\{T_w\})_{w \in R_x}$ (i.e., "shares" of $s = 0$ and "shares" of $s = 1$) are identical. Thus, a poly-time adversary cannot distinguish with non-negligible advantage between shares of $s = 0$ and shares of $s = 1$. □

We next plug-in the constructions of pPRG in Theorem 4.1.

Corollary 4.2. *For every graph $G = (V, E)$, there are a SCSSs with the following share size and assumptions:*

- *Share size $\log |V| \cdot \text{poly}(\lambda)$ under the polynomial RSA assumption.*
- *Share size $|V|^{1/2} \cdot \text{poly}(\lambda)$ under the polynomial LWE assumption.*
- *Share size $|V|^{2/3} \cdot \text{poly}(\lambda)$ under the polynomial DDH assumption.*

Proof. The SCSS from the RSA assumption follows immediately from plugging-in Theorem 3.6 in Theorem 4.1. For the construction from the DDH assumption, we use Corollary 3.15 with $m = |V|$ and $= 2/3$, i.e., a pPRG with key-size $|V|^{2/3} \text{poly}\,\lambda$ and public parameter of length $|V|^{4/3} \text{poly}\,\lambda$. We want to construct a SCSS without public information. We observe that in the construction for bipartite graph in the proof of Theorem 4.1, the share of $v_i \in L$ is $s \oplus c_i$ and an edge (v_i, w) reconstructs s by computing c_i. By Corollary 3.15, this can be done by using $|V|^{2/3} \text{poly}(\lambda)$ bits from the public parameter. Thus, we modify the SCSS by adding this part from the public parameter to the share of v_i. The construction from the LWE assumption is done in a similar way using Corollary A.4 with with $m = |V|$ and $= 1/2$. □

Remark 4.3. *In the construction of the SCSS for bipartite graphs from the RSA assumption, the share of each vertex $v_i \in L$ is one bit, while the share size of each vertex $w \in R$ is $\log |V| \text{poly}(\lambda)$. It can be shown that there exists a bipartite graph G such that in any information-theoretic secret-sharing schemes for G in which the share of each vertex $v_i \in L$ is one bit, the share size of the vertices in R is $\Omega(|V|)$. Thus, we get another example of separation between computational and information-theoretic secret-sharing schemes.*

Remark 4.4. *We can consider a "multi-secret" generalization of secret-sharing schemes for a* bipartite *graph $G = (L = \{v_1, \dots, v_m\}, R, E)$ where every vertex v_i has a secret $s_i \in \{0, 1\}$ and for an input x:*

- *The parties in S_x can reconstruct all secrets s_i such that $v_i \in S_x \cap L$ and there is $w \in S_x \cap R$ such that $(v_i, w) \in E$, and*
- *Let $T^* = \{i \in [m] : (v_i \notin S_x \cap L) \vee (\forall_{w \in S_x \cap R} (v_i, w) \notin E)\}$ (i.e., all the secrets that x cannot learn from the previous item). The parties in S_x cannot distinguish if the secrets of the vertices T^* are $(s_j)_{i \in T^*}$ or $(s'_j)_{i \in T^*}$ for every two vectors of secrets $(s_j)_{i \in T^*}$ and $(s'_j)_{i \in T^*}$.*

To share the secrets $s_1, \dots, s_m \in \{0, 1\}$ in a bipartite graph secret-sharing scheme, we modify the above secret-sharing for bipartite graphs by giving every vertex $v_i \in L$ the share $s_i \oplus c_i$. The share of $w \in R$ is not changed.

4.2 SCSS for CNF Formulas and Truth Tables

We next describe an efficient SCSS for functions represented as monotone CNF formulas. Let $\varphi = \wedge_{j=1}^{m} C_i$ be a monotone CNF formula over the variables $y_1, \ldots, y_n$ with m monotone clauses, where each clause is a disjunction over a subset of the variables. We first recall a variant of the information-theoretic secret-sharing scheme of [ISN87] for the CNF formula φ: to share a one bit secret s, the dealer chooses m random bits $r_1, \ldots, r_m$, one for each clause, computes $\mathsf{sh}_0 = r_1 \oplus \cdots \oplus r_m \oplus s$ and gives the i^{th} party the share $\mathsf{sh}_0, (r_j)_{\{j : y_i \text{ is in the clause } C_j\}}$. As a result, a coalition of parties can recover the secret if and only if they have at least one representative in each clause. In this scheme, the share of a party can be as large as $m + 1$. To get a SCSS with short shares, we generate $c_1, \ldots, c_m$ as the output of a robust pPRG, and the share of a party is a projective key for the bits it should learn.

Theorem 4.5 (Theorem 1.4 restated). *Assume that there is a $t(\lambda)$-robust pPRG in which the size of the projective keys is $\kappa(\lambda, m)$ and the size of the public parameters is $\kappa_0(\lambda, m)$, where m is the output length of the generator. Then, there is a $\mathrm{poly}(t(\lambda))$-secure SCSS for monotone CNF formulas with share size $\kappa(\lambda, m)$ and public information size $\kappa_0(\lambda, m)$, where m is the number of clauses in the CNF formula.*

The theorem states that the security loss in the reduction is polynomial and so the reduction preserves both polynomial and sub-exponential hardness. This is true for all the constructions in the paper, though we make this explicit only in some of the cases (where it is beneficial to consider the setting of sub-exponential hardness).

Proof. The sharing algorithm $\mathrm{CSS.S{\scriptsize HARE}}(1^\lambda, \varphi = \wedge_{j=1}^{m} C_i, s)$ of the secret-sharing scheme works as follows.

- Compute $(\mathsf{params}, \mathsf{msk}) \leftarrow \mathrm{pPRG.S{\scriptsize ETUP}}(1^\lambda, 1^m)$.
- Compute $(c_1, \ldots, c_m) \leftarrow \mathrm{pPRG.E{\scriptsize VAL}}(\mathsf{params}, \mathsf{msk})$.
- The public information sh_0 is $(\mathsf{params}, c_1 \oplus \cdots \oplus c_m \oplus s)$.
- For an index $i \in [n]$, let $T_i = \{j : y_i \text{ is in the clause } C_j\}$. The share sh_i is $a\{T_i\} \leftarrow \mathrm{pPRG.K{\scriptsize EY}G{\scriptsize EN}}(\mathsf{params}, \mathsf{msk}, T_i)$.

For the correctness of the scheme, consider $x \in \{0,1\}^n$ such that $\varphi(x) = 1$, that is, for every clause C_j in φ there exists a variable y_i in C_j such that $x_i = 1$, i.e., $j \in T_i$ and c_j can be computed by the key $a\{T_i\}$ held by the parties in S_x. Thus, the parties in S_x can compute $c_1, \ldots, c_m$ and reconstruct s from them and sh_0. The security for an input x and a monotone formula φ chosen by the adversary such that $\varphi(x) = 0$ follows from the fact that there is at least one clause C_j that is not satisfied by x, i.e., $x_i = 0$ for every y_i in C_j. This implies that given the projective keys held by S_x, the bit c_j is indistinguishable in polynomial time from a random bit and an adversary cannot distinguish with non-negligible probability between shares of $s = 0$ and shares of $s = 1$. □

Given a truth table representing a monotone function $f : \{0,1\}^n \to \{0,1\}$, we can represent f by a monotone CNF formula φ_f with $m \leq 2^n$ clauses and execute the above SCSS for φ_f. We assume sub-exponential security of the pPRG, i.e., it is $2^{(\lambda_{\mathrm{p}})^{\delta}}$-secure for some constant $\delta > 0$ (where λ_{p} is the security parameter of the pPRG). We take the security parameter of the pPRG as $\lambda_{\mathrm{p}} = \lambda^{2/\delta}$, thus $2^{(\lambda_{\mathrm{p}})^{\delta}} = 2^{\lambda^2}$ and we get exponential hardness of the SCSS.

Corollary 4.6. *Assume that there exists a $2^{(\lambda_{\mathrm{p}})^{\delta}}$-robust pPRG for some constant $\delta > 0$ in which the size of the projective keys is $\kappa(\lambda_{\mathrm{p}}, m)$, where m is the size of the output of the generator. Then, there exists a SCSS for functions $f : \{0,1\}^n \to \{0,1\}$, represented by a truth tables of size $N = 2^n$, with share size $\kappa(\lambda^{2/\delta}, N)$ that is 2^{λ^2}-secure. The sharing and reconstruction algorithms of the SCSS run in time $\tilde{O}(N) \cdot \mathrm{poly}(\lambda)$.*

Plugging in the pPRG constructed under the sub-exponential RSA assumption in Theorem 3.6 with projective key size $\mathrm{poly}(\lambda_{\mathrm{p}}, \log m)$, we obtain the following corollary.

Corollary 4.7. *Under the sub-exponential RSA assumption, there exists a SCSS for functions $f : \{0,1\}^n \to \{0,1\}$, represented by a truth table of size $N = 2^n$ with share size $\mathrm{poly}(\lambda, \log N) = \mathrm{poly}(\lambda, n)$ that is 2^{λ^2}-secure. The sharing and reconstruction algorithms of the SCSS run in time $\tilde{O}(N) \cdot \mathrm{poly}(\lambda)$.*

Remark 4.8. *For every function $f : \{0,1\}^n \to \{0,1\}$, we construct a SCSS with short share size, however the running time of the sharing and reconstruction algorithms is quasi-linear in the representation size N, i.e., it is exponential in n. This is consistent with our definitions as the size of the truth table (i.e., the "program") is $N = 2^n$. However, this exponential-time run time stems from deeper reasons. The sharing algorithm has to "know" all minimal inputs such that $f(x) = 1$ otherwise it will not be able to generate shares such that only inputs x such that $f(x) = 1$ can reconstruct the secret. If we consider functions such that all minimal such inputs have weight exactly $n/2$, there are $\Omega(2^n/\sqrt{n})$ possible inputs and the running time has to be exponential.*

The reconstruction algorithm could ignore the program P and reconstruct the secret from x and the shares, thus possibly avoiding the exponential-time reconstruction. However, Larsen and Simkin [LS20] proved that for almost all functions $f : \{0,1\}^n \to \{0,1\}$ in any secret-sharing scheme for f either the share size is exponential or the running-time of the reconstruction algorithm is exponential (i.e., the circuit size computing the reconstruction is exponential).[8] *Thus, our scheme with polynomial share size has optimal running time for the sharing and reconstruction algorithms.*

4.3 SCSS for Monotone Circuits

Yao [Yao89, VNS+03] has shown how to construct a SCSS for monotone circuits of bounded fan-in with share size that is linear in the circuit size and the security parameter. For the special case of

[8]The statement of the result of [LS20] only mentions information-theoretic secret-sharing schemes, however, their results also apply to computational secret-sharing schemes as (in a special case of their result) they only require that for every input x such that $f(x) = 1$ there exists a sub-exponential circuit that reconstructs the secret and for every input x such that $f(x) = 0$ there does not exist a sub-exponential circuit that distinguishes with probability at least $1/4$ between shares of 0 and shares of 1. The first property follows from the assumption that there is a SCSS with sub-exponential time reconstruction algorithm and the second follows from the exponential-security of the SCSS.

monotone circuits over OR and AND gates of unbounded fan-in (hereafter referred to as *AND-OR circuits*), this gives a complexity that grows linearly in the number of wires (and the security parameter). We use a pPRG to get a construction whose complexity grows linearly with the number of *gates* which, for general circuits, yields a quadratic improvement. In fact, we can handle OR gates "for free" and pay only for AND gates. (This yields a strict generalization of the CNF construction.)

Remark 4.9 (Block-pPRG). *Instead of using a pPRG directly, it will be convenient to employ a* block pPRG *which forms a natural generalization of a pPRG. Specifically, in a block pPRG we think of the pseudorandom output as a sequence $(c_1, \ldots, c_m)$ of m blocks, where each block c_i is of length λ. A projective key $a\{T\}$ of a set $T \subseteq [m]$ should allow computing all the blocks c_i for which $i \in T$. The security and robustness games are defined naturally where the only difference is that c_0 is sampled uniformly from $(\{0,1\}^\lambda)^{|\overline{T}|}$.*

Any (robust) pPRG with an output length of $m' = m\lambda$ can be viewed as a (robust) block pPRG of length m by parsing the outputs as blocks and by setting the projective key of T to $a\{T'\}$, where T' is the set of all location that fall inside the blocks whose index is in T. Alternatively, one can simply concatenate λ-independent copies of a pPRG of output length m and set the i^{th} bit of the j^{th} block to be the j^{th} output bit of the i^{th} copy. This transformation preserves robustness and increases the key size and the size of the public parameters by a factor of λ, and therefore succinctness and strong succinctness are preserved. Finally, we note that concrete constructions (e.g., those based on RSA, DDH) can be easily modified to obtain block-pPRGs directly at a minor cost (e.g., in RSA, one can extract many hard-core bits from the same element).

We prove the following theorem.

Theorem 4.10 (Theorem 1.5 restated). *Assume that there is a robust block-pPRG in which the length of the projective keys is $\kappa(\lambda, m)$ and the length of the public parameters is $\kappa_0(\lambda, m)$, where m is the output length (number of blocks) of the generator and each block is of length λ. Then, there is a SCSS for AND-OR circuits whose share size is $\kappa(\lambda, m)$ and its public information size is $\kappa_0(\lambda, m) + m_\wedge \cdot \lambda + 1$, where m is the number of gates and $m_\wedge$ is the number of AND-gates.*

Remark 4.11 (Extensions). *The theorem and its proof extend naturally to the sub-exponential setting. In particular, if the pPRG is sub-exponentially secure so is the resulting SCSS. One can also extend the construction in a natural way to the case of multi-output circuits and get a multi-output secret-sharing scheme in which each secret is associated with an output gate of the circuit and is released if and only if the gate is satisfied. A refined analysis shows that the share size in this case is $\kappa(\lambda, m_\vee) + \lambda$ and its public information size is $\kappa_0(\lambda, m_\vee) + m_\wedge \cdot \lambda + \ell$, where $m_\vee$ is the number of OR gates, $m_\wedge$ is the number of non-output AND gates, and ℓ is the number of outputs.*

Remark 4.12 (Special cases). *Consider the case where the circuit C is a CNF formula. In this case, we have a single AND-gate which is the output gate and so, by the refined analysis, we essentially get the pPRG-based CNF construction up to an additive cost of λ in the sharing size. In fact, a closer look at the construction below shows that, for CNF, one can* *get exactly the same parameters and also replace the block-pPRG with a robust pPRG, just like in Theorem 4.5.*

Next, consider the case where C is a DNF formula. In this case, we can remove the top OR gate, view each term as an output gate, and associate the same secret s to all the outputs. This allows us to realize an ℓ-term DNF with a share size of $\kappa(\lambda, m_\vee) + \lambda$ and public information size of $\kappa_0(\lambda, m_\vee) + m_\wedge \cdot \lambda + \ell$. Since there are neither OR gate nor non-output AND gates, i.e., $m_\vee = m_\wedge = 0$, we get shares of size λ and public information of size ℓ. Moreover, since $m_\vee = 0$, we do not need pPRGs and can rely solely on PRGs, and we derive Theorem 1.9 (that will be proved directly as Theorem 6.1).

4.3.1 Proof of Theorem 4.10

Notation and conventions. Let C be an AND-OR circuit with variables $y_1, \dots, y_n$. We will assume, w.l.o.g., that all the outgoing wires of an OR gate are connected as incoming wires to AND gates, and vice-versa, that all the outgoing wires of an AND gate are connected as incoming wires to OR gates. If this is not the case and, say, an OR-gate g has an outgoing wire that enters an OR-gate h, we can shortcut the gate g, i.e., duplicate all the input wires of g and connect them directly to h. If the gate g has no additional outgoing wires it can be deleted. (If g is connected to some AND gate, we keep it in the circuit.) This transformation does not increase the number of gates. Furthermore, we assume, for simplicity, that each input gate is only connected to OR gates. This can be achieved by adding for every y_i an OR gate with fan-in 1. (Consequently, the number of OR gates is increased by at most n, but as we will see, the theorem statement regarding the complexity of the scheme remains as stated.) We also assume that gates are numbered from 1 to m according to some topological order and that the first n gates are input gates that correspond to the inputs $y_1, \dots, y_n$. We write $i \to j$ if an output of the i^{th} gate is being fed to the j^{th} gate as an input.

Overview. At a high level, we allocate to each gate i a key K_i and make sure that a set of authorized parties $x \in \{0,1\}^n$ will be able to learn the keys of the gates that are satisfied by x and learn nothing about the other keys. The keys of OR gates will be pseudorandom blocks from the output of the pPRG, whereas the keys of the AND gates will be the projective keys. We will also publish some public ciphertexts that allow us to move from an OR layer to an AND layer. In addition to $\text{pPRG} = (\text{pPRG.Setup}, \text{pPRG.KeyGen}, \text{pPRG.Eval})$, our construction employs a (standard) output-variable pseudorandom generator G whose existence follows from the existence of any one-way function [HILL99] and therefore also from the existence of a pPRG.

The construction. The sharing algorithm $\text{CSS.Share}(1^\lambda, C, s)$ of the secret-sharing scheme works as follows.

- Let $m_\vee$ denote the number of OR gates in C and let m denote the number of gates in C. Let $\kappa = \kappa(\lambda, m_\vee)$ be the length of the projective keys of the pPRG scheme. Let $\ell = m \cdot \kappa$ and let us set the output length of the PRG G to ℓ, we abuse notation and denote the mapping $G(\cdot, 1^\ell) : \{0,1\}^\lambda \to \{0,1\}^\ell$ by G.[9] We parse the output of G as composed of m blocks, each

[9]We restrict ourselves to polynomial adversaries and therefore assume that the output length ℓ is polynomial in λ.

of length κ, and for a seed $a \in \{0,1\}^\lambda$ and index $i \in [m]$, we let $G(a)[i] \in \{0,1\}^\kappa$ denote the i^{th} block of G. Compute $(\mathsf{params}, \mathsf{msk}) \leftarrow \text{pPRG.}\textsc{Setup}(1^\lambda, 1^{m_\vee})$, and $(c_1, \ldots, c_m) \leftarrow \text{pPRG.}\textsc{Eval}(\mathsf{params}, \mathsf{msk})$, where c_i is a block of length λ. Place params as part of the public information sh_0.

- Set the key K_i of the i^{th} gate as follows:
  - For an OR gate set $K_i \leftarrow c_i \in \{0,1\}^\lambda$ to be the i^{th} pseudorandom block of the pPRG.
  - For an AND gate and an input gate, set $K_i \in \{0,1\}^\kappa$ to be the projective key $a\{T_i\} \leftarrow \text{pPRG.}\textsc{KeyGen}(\mathsf{params}, \mathsf{msk}, T_i)$ that corresponds to the set $T_i = \{j : i \to j\}$ of all i's out-neighbors. If this gate is an output gate, set $K_i = s$, where s is the secret. If this is an input gate, set the i^{th} share to $\mathsf{sh}_i = K_i = a\{T_i\}$.
- Public information: for every non-output AND gate i, we publish, as part of the public information sh_0, the ciphertext $E_i = K_i \oplus R_i$, which is viewed as an encryption of K_i under the mask R_i, defined via
$$R_i = \bigoplus_{j \to i} G(K_j)[i].$$
If the AND gate i is an output gate the plaintext K_i is a single bit, and so we use only the first bit of the mask R_i. If the output gate m is an OR gate, we also publish a one-bit ciphertext $E_m = s \oplus K_m[1]$, where $K_m[1]$ is the first bit of the key K_m of the m^{th} gate. In any case, the ciphertext E_m is a single-bit ciphertext.

The reconstruction algorithm CSS.$\textsc{Recon}$ works as follows. Let $x \in \{0,1\}^n$ be an input that is accepted by the circuit C. We traverse the circuit from the inputs to the output, and recover the key K_i of each gate i that is satisfied by x (i.e., the gate is evaluated to 1 under the assignment x). For an input gate, K_i is given as part of the shares of x. For an OR gate, the key $K_i = c_i$ can be recovered based on the key K_j of the first gate j that is satisfied and whose outgoing wire enters i, i.e., $j \to i$. Indeed, j is either an input gate or an AND gate, and in any case its key K_j, which was already recovered, consists of a projective key $a\{T_j\}$ for a set T_j that contains i. For AND gate, the key K_i can be recovered by XOR-ing the ciphertext E_i with the mask R_i. This mask can be computed based on all the keys $\{K_j : j \to i\}$ that were already recovered (since $j < i$ and since all the gates $j : j \to i$ must be satisfied by x). After we reconstruct the key K_m of the output gate, we can recover the secret by decrypting the ciphertext E_m.

Correctness, Complexity, and Security. It can be proved (e.g., by induction on i) that the key of the i^{th} satisfied gate is recovered correctly, thus correctness holds. The share size is $\kappa(\lambda, m_\vee)$ and its public information size is at most $\kappa_0(\lambda, m_\vee) + m_\wedge \cdot \lambda + 1$, where $m_\vee$ is the number of OR gates and $m_\wedge$ is the number of AND-gates. Since we increased the original number of OR gates by at most n (in order to make the assumption that inputs are only connected to OR gates), it holds that $m_\vee$ is upper-bounded by m the number of gates in the original circuit, and so the

However, the construction naturally generalizes to sub-exponential adversaries, assuming that the pPRG and the PRG are sub-exponentially secure. (Again, the existence of the latter follows from the existence of the former.)

theorem's statement holds. Security is proved via the following claim, which completes the proof of Theorem 4.10.

Claim 4.13. *The above construction forms a secure SCSS.*

Proof. For the security of the scheme, consider an input x and a circuit C chosen by the adversary $\mathcal{A}$ such that $C(x) = 0$. Let us assume that the adversary wins in the security game with a probability of $0.5 + \varepsilon$. For every $1 \leq h \leq m + 1$, consider the hybrid game H_h in which the secret sharing that is given to the adversary is defined as in $\mathsf{CSS.Share}(1^\lambda, C, s)$ except that for every $i < h$ if the i^{th} gate is an unsatisfied AND/output gate, we sample the ciphertext E_i independently at random and if the i^{th} gate is an unsatisfied OR gate we sample its key K_i by a truly random string. Let us refer to this distribution of the shares as $\mathsf{CSS.Share}(1^\lambda, C, s, x, h)$. Let p_h denote the winning probability of the adversary in the h^{th} game. Clearly, H_0 corresponds to the standard security game and so $p_0 = 0.5 + \varepsilon$, by assumption. Also, in H_m the distribution $\mathsf{CSS.Share}(1^\lambda, C, s, x, m)$ that is given to the adversary is statistically independent of the secret s and so $p_m = 0.5$. Therefore, it suffices to show that, for every $h \in [m]$, $\varepsilon_h := \|p_h - p_{h+1}\|$ is negligible. Observe that the difference is non-zero only if the h^{th} gate is an unsatisfied gate which is either an AND gate or an OR gate (but not an input gate). We consider the following 2 cases.

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 K_j of this gate is chosen at random in both hybrids, so we can use the distinguisher to break the pseudorandomness of $G(K_j)$ as follows. Given m blocks $z = (z_1, \ldots, z_m)$, where $z_i \in \{0,1\}^\kappa$ we distinguish between the case where

$$z_i \leftarrow_U \{0,1\}^\kappa,\ \forall i \leq h-1, \quad \text{and} \quad z_i \leftarrow G(K_j)[i], \forall i > h-1, \qquad \text{where } K_j \leftarrow_U \{0,1\}^\lambda \tag{5}$$

and the case where

$$z_i \leftarrow_U \{0,1\}^\kappa,\ \forall i \leq h+1, \quad \text{and} \quad z_i \leftarrow G(K_j)[i], \forall i > h+1, \qquad \text{where } K_j \leftarrow_U \{0,1\}^\lambda, \tag{6}$$

by sampling the shares exactly as in $\mathsf{CSS.Share}(1^\lambda, C, s, x, h-1)$ except that, for each $i \in [m]$, we replace the block $G(K_j)[i]$ with the block z_i. It is not hard to verify that if z is distributed as in (5) the shares are distributed like in $\mathsf{CSS.Share}(1^\lambda, C, s, x, h-1)$ and if z is distributed according to (6) we get $\mathsf{CSS.Share}(1^\lambda, C, s, x, h)$. We can therefore distinguish between the two with an advantage of ε_h, which by a standard hybrid argument allows us to break pseudorandomness of G (by predicting a single bit) with an advantage of ε_h/κ. We conclude that ε_h must be negligible.

Case 2: the h^{th} gate is an OR gate. In this case, all the incoming wires must be connected to unsatisfied gates, and we can use the adversary to break the pPRG with an advantage of ε_h as follows. Initialize the game with parameters $(1^\lambda, 1^m)$ and ask for the projective keys of all the sets T's that are used by $\mathsf{CSS.Share}(1^\lambda, C, s)$ except for sets that contain h. Given the projective keys, and the challenge that contains the block c_j, we sample the shares as in $\mathsf{CSS.Share}(1^\lambda, C, s, x, h-1)$ except that the key K_h is set to c_h.[10] In addition, we use the projective keys to generate all other

[10]Note that the projective keys for sets K that contain h are not needed in order to produce the output of $\mathsf{CSS.Share}^i(1^\lambda, C, s, x)$ since such keys can appear either as part of a sharing K_i of an input gate $i \to h$, or as part of a ciphertext E_i of an AND gate $i \to h$. However, since h is an unsatisfied OR gate, any such input gate i (resp., AND gate i) must also be unsatisfied, and its share is omitted (resp., its ciphertext is just uniform).

pseudorandom blocks c_r for $r \geq h$ that are consumed by the sharing algorithm. This means that, when c_j is pseudorandom the shares are distributed just like in CSS.$\textsc{Share}(1^\lambda, C, s, x, h-1)$ and if c_j is random the shares are distributed just like in CSS.$\textsc{Share}(1^\lambda, C, s, x, h)$. Hence, we can win the pPRG game with advantage ε_h, by outputting 1 whenever $\mathcal{A}$ guesses s. We conclude that ε_h must be negligible. This completes the proof of the claim. □

5 SCSS for Partite Functions and Forbidden Graphs from OWFs

In this section, we construct an efficient SCSS for a class of functions, called partite functions, under the most basic cryptographic assumption that one-way functions exist. Partite functions $f : \{0,1\}^{2n} \to \{0,1\}$ are monotone functions that encode every (possibly non-monotone) function $g : \{0,1\}^n \to \{0,1\}$ by essentially allocating a variable to each original literal (see Definition 5.1). Secret-sharing schemes for partite functions are closely related to so-called multi-server fully-decomposable CDS protocols [GIKM00] (which, in turn, are closely related to partial garbling schemes [IW14] or privacy-free garbling schemes [FNO15]), and have many interesting applications. In particular, they are used in the best-known constructions of information-theoretic secret sharing (ITSS) for arbitrary access structures [LV18, ABF+19, ABNP20, AN21]. By [LVW17], every partite function can be realized by an ITSS with share size $2^{O(\sqrt{n}\log n)}$. We construct a SCSS for partite functions whose share size is λ.

Definition 5.1 (Partite Functions). *We encode any input $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ by an input $y = (y_1, \ldots, y_{2n}) \in \{0,1\}^{2n}$, where if $x_i = 0$ then $y_{2i-1} = 0$ and $y_{2i} = 1$ and if $x_i = 1$ then $y_{2i-1} = 1$ and $y_{2i} = 0$ (note that such y has weight exactly n). We define the partite function $f : \{0,1\}^{2n} \to \{0,1\}$ of a function $g : \{0,1\}^n \to \{0,1\}$ as a monotone function whose minterms are:*

- *All inputs y of weight 2 such that $y_{2i-1} = y_{2i} = 1$, for some i.*
- *All inputs y that encode an input x such that $g(x) = 1$.*

We say that f is a partite function if it is a partite function of some function g.

Put differently, f is a simple monotone function that "agrees" with g, i.e., $f(y) = g(x)$ for every x and every y that encodes x. To make the function fully-defined, the first condition is added, and as a result, if the weight of y is $n+1$ then $f(y) = 1$.

Theorem 5.2 (Theorem 1.7 restated). *Assuming $t(\lambda)$-secure one-way functions exist, for all partite functions $f : \{0,1\}^{2n} \to \{0,1\}$, for some n, represented by truth tables of size $N = 2^n$, there exists a $\mathrm{poly}(t(\lambda))$-secure SCSS with share size $\lambda + O(1)$. The sharing and reconstruction algorithms of the SCSS run in time $N \cdot \mathrm{poly}(\lambda)$.*

Proof. Given a partite function F, it suffices to realize the partial function f that is defined only over inputs of the form $\{01, 10\}^n$. We can then write $F(y)$ as

$$\left(f(y) \wedge \bigwedge_{i\in[n]} (y_{2i-1} \vee y_{2i}) \right) \vee \left(\bigvee_{i\in[n]} (y_{2i-1} \wedge y_{2i}) \right).$$

By using standard closure properties of secret sharing, this allows us to realize F with the same share size as f with $O(1)$ additive cost (concretely, with 2 additional bits per party). From here on we focus on partial partite functions.

By [HILL99], t-secure one-way functions imply $\mathrm{poly}(t)$-secure PRGs. The description of the SCSS construction is recursive. If $n = 2$ (i.e., $f : \{0,1\}^2 \to \{0,1\}$) then $\mathsf{sh}_1 = s$ if $f(10) = 1$ and $\mathsf{sh}_1 = 0$ otherwise. Similarly, $\mathsf{sh}_2 = s$ if $f(01) = 1$ and $\mathsf{sh}_2 = 0$ otherwise.

For the recursive step, we construct a SCSS for a partial partite function $f : \{0,1\}^{2n} \to \{0,1\}$ whose randomness complexity is $2\lambda + 1$ and its share complexity is at most $\lambda + 1$. Define $f_1, f_2 : \{0,1\}^{2n-2} \to \{0,1\}$ as the partite functions, where $f_1(y) = f(y \circ 10)$ and $f_2(y) = f(y \circ 01)$ (where $\circ$ is concatenation of strings). Thus,

$$f(y_1, \ldots, y_{2n}) = (y_{2n-1} \wedge f_1(y_1, \ldots, y_{2n-2})) \vee (y_{2n} \wedge f_2(y_1, \ldots, y_{2n-2})).$$

Let $\mathrm{CSS}.\textsc{Share}(1^\lambda, f_b, s)$ be the sharing algorithm for the function f_b for $b \in \{1,2\}$ that exists by our recursive construction. The algorithm $\mathrm{CSS}.\textsc{Share}(1^\lambda, f, s)$ for the function f is constructed as follows:

- Choose two uniformly distributed seeds $a_1, a_2 \in_U \{0,1\}^\lambda$ for a pseudorandom generator $\mathrm{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+1}$, sample uniformly at random a bit $s_1 \in_U \{0,1\}$, and set $s_2 = s \oplus s_2$.
- For $b \in \{1,2\}$, let $r_b = \mathrm{PRG}(a_b)$ and $\mathsf{sh}^b_1, \ldots, \mathsf{sh}^b_{2n-2} \leftarrow \mathrm{CSS}.\textsc{Share}(1^\lambda, f_b, s_b; r_b)$. That is, r_b is used by the randomized algorithm $\mathrm{CSS}.\textsc{Share}$ for the function f_b as its random string.
- For $1 \le i \le 2n-2$, let $\mathsf{sh}_i = \mathsf{sh}^1_i \oplus \mathsf{sh}^2_i$.
- $\mathsf{sh}_{2n-1} \leftarrow (s_2, a_2)$ and $\mathsf{sh}_{2n} \leftarrow (s_1, a_1)$.

Correctness. Let $y \in \{01,10\}^n$ be such that $f(y) = 1$. If $y_{2n-1} = 1$, then $f_1(y_1, \ldots, y_{2n-2}) = 1$ (because $f(y) = 1$). In this case, $\mathrm{CSS}.\textsc{Recon}(1^\lambda, f, (\mathsf{sh}_i)_{i \in S_y})$ does the following:

- Compute $\mathsf{sh}^2_1, \ldots, \mathsf{sh}^2_{2n-2} \leftarrow \mathrm{CSS}.\textsc{Share}(1^\lambda, f_2, s_2; r_2)$ using the secret s_2 and the random string $r_2 = \mathrm{PRG}(a_2)$, where s_2 and a_2 are retrieved from the share sh_{2n-1}.
- Compute $\mathsf{sh}^1_i \leftarrow \mathsf{sh}_i \oplus \mathsf{sh}^2_i$ for every $i \in S_{y_1,\ldots,y_{2n-2}}$ (where all shares sh^2_i were computed in the previous step, and the needed shares sh_i are in the input) and
$$s_1 \leftarrow \mathrm{CSS}.\textsc{Recon}(1^\lambda, f_1, (\mathsf{sh}^1_i)_{i \in S_{y_1,\ldots,y_{2n-2}}}).$$
- Reconstructs s from s_1 and s_2 (that was already retrieved from sh_{2n-1}).

The case $y_{2n} = 1$ is analogous.

Security. Let $\mathcal{A}$ be a non-uniform $t(\lambda)$-time adversary that breaks the SCSS with probability $0.5+\varepsilon(\lambda)$, we will show how to use $\mathcal{A}$ to break the security of the PRG. Fix the security parameter λ, let $t=t(\lambda)$, $\varepsilon=\varepsilon(\lambda)$, and let f be the partite function with input length $2n$ and $y \in \{01,10\}^n$ be the input chosen by $\mathcal{A}$ such that $f(y)=0$. Let us arrange the recursive calls of the sharing algorithm in a binary tree and label each call in the i^{th} level of the recursion by a string $z \in \{01,10\}^i$ such that the z^{th} call corresponds to the partial function $f|_z$ that takes $x \in \{0,1\}^{2n-2i}$ to $f(x,z)$. We let s_z denote the secret that is shared in the z^{th} call, let r_z denote the random tape that is being used in the z^{th} call, let a_z denote the seed that generates r_z, and let $\mathsf{sh}^z=(\mathsf{sh}^z_1,\dots,\mathsf{sh}^z_{2n-|z|}) \leftarrow \text{CSS.S}\textsc{hare}(1^\lambda, f|_z, s_z; r_z)$ denote the shares that are sampled in the z^{th} call. Denote by $z_j := y[2n-2j+1:2n]$ the $2j$-bit suffix of the string y. Consider the path that is labeled by $z_0,\dots,z_{n-1}$. The main observation is for each vertex z in the path it holds that (1) the partial function $f|_z$ evaluates to zero on the corresponding prefix of y; and (2) the seed a_z that generates the random tape of the z^{th} call is not given to the adversary (it appears in a share of a party that does not participate in S_y). As a result, we can replace these pseudorandom tapes with random tapes and argue that the corresponding scheme perfectly hides the secret and is computationally indistinguishable from the original scheme. Details follow.

For every $0 \le h < n-1$, consider the hybrid game H_h in which the shares that are delivered to $\mathcal{A}$ in the security game are modified as follows. For every $j \le h$, when making the recursive call that is labeled by z_j, we use a fresh random string r_{z_j} that is sampled uniformly at random and not via the PRG. Observe that the first hybrid H_0 corresponds to the standard security game. We also claim that H_{n-1} is statistically independent of the secret.

Claim 5.3. *The distribution that the adversary sees in the game H_{n-1} is statistically independent of the secret s.*

Proof. For $0 \le j \le n$, let $W_j=\{i \le 2n-2j : y_i = 1\}$ denote the parties that participate in (the characteristic set of) y among the first $2n-2j$ parties. We denote the shares that are sampled in the z_j^{th} call by

$$\mathsf{sh}^{z_j}=(\mathsf{sh}^{z_j}_1,\dots,\mathsf{sh}^{z_j}_{2n-|z|}) \leftarrow \text{CSS.S}\textsc{hare}(1^\lambda, f|_{z_j}, s_{z_j}; r_{z_j}),$$

where r_{z_j} is uniform (since we are in the hybrid H_{n-1}). We will prove that, in H_{n-1}, for every $j \le n$, it holds that the secret s_{z_j} that is secret-shared in the z_j^{th} call is distributed independently from $\mathsf{sh}^{z_j}_{W_j}=(\mathsf{sh}^{z_j}_i)_{i\in W_j}$. Thus the claim follows from the case of $j=n$.

The proof is by induction on j. The base case $j=1$ trivially follows from the information-theoretic security of the base case of the construction. For the induction step, assume that $(y_{2j-1},y_{2j})=10$ (the other case is symmetric). Then, $\mathsf{sh}^{z_j}_{W_j}$ can be computed based on $\mathsf{sh}^{z_j}_{2j-1}=(s_2,a_2)$ and the shares $\mathsf{sh}^{z_{j-1}}_{W_{j-1}}$ and $(\mathsf{sh}^2_i)_{i\in W_{j-1}}$. Here $s_2=s_{z_j}\oplus s_{z_{j-1}}$, the string a_2 is a fresh random seed, and $(\mathsf{sh}^2_i)_{i\in W_{j-1}}$ can be efficiently sampled based on s_2 and a_2. By the induction hypothesis, the random variable $\mathsf{sh}^{z_{j-1}}_{W_{j-1}}$ is statistically independent of the random variable $s_{z_{j-1}}$ and so the claim follows. □

We conclude that the winning probability in H_n is 0.5, and so there must be a pair of neighboring hybrids that can be distinguished with an advantage of $\varepsilon'=\varepsilon/n$. We show that this leads to an attack on the PRG.

Claim 5.4. *Suppose that H_{h-1} and H_h can be distinguished with advantage ε' by a t-size circuit. Then the PRG can be broken with advantage ε' by a $t' = t + N\operatorname{poly}(\lambda)$-size adversary.*

Proof. Given a PRG challenge r_{z_h} sample the hybrid H_{h-1} except that the random tape that corresponds to z_h is set to r_{z_h}. Note that we do not need the seed for z_h since it only appears as part of the share of the party $i \in \{n-2h-1, n-2h\}$ for which $y_i = 0$. The resulting distribution corresponds to H_{h-1} if r_{z_h} is pseudorandom and to H_h if r_{z_h} is uniform. Thus we can apply the distinguisher to break the PRG. The computational overhead of the reduction is $N\operatorname{poly}(\lambda)$ (which is the complexity of sampling H_{h-1}). □

Since the adversary specifies the truth table of f in time t it holds that $t \geq N$, also $n = \log N$ and, by assumption $t > \lambda$. Therefore $t' = \operatorname{poly}(t)$ and $\varepsilon' = \varepsilon/n > \varepsilon/\log t$. We conclude that the PRG can be broken by a distinguisher with complexity $t' = \operatorname{poly}(t,\lambda)$ and advantage of $\varepsilon' > \varepsilon/\log t > \varepsilon/\operatorname{poly}(t)$ as required. This completes the proof of the theorem. □

5.1 SCSS for Forbidden Graphs

Secret-sharing schemes for *forbidden graphs* [SS97] are similar to secret sharing for graphs (defined in Section 4.1), however the security requirement is relaxed. Given an undirected graph $G = (V, E)$, we view the parties as the vertices in V, and the authorized sets are those sets that contain at least one edge from E and *all sets of size at least* 3. We will mainly consider secret-sharing schemes for *bipartite* graphs, namely, where V consists of two sets L, R and all edges connect vertices from both sets, i.e. $E \subseteq L \times R$. Secret-sharing schemes for forbidden bipartite graphs are basically equivalent to 2-server conditional disclosure of secrets (CDS) protocols, defined in [GIKM00]. Furthermore, they imply secret-sharing schemes for arbitrary forbidden graphs with a $\log|V|$ multiplicative overhead [BIKK14].

We show that a SCSS for partite functions imply a SCSS for bipartite forbidden graphs (and therefor for arbitrary forbidden graphs), i.e., a 2-server CDS protocol for an arbitrary function. Given a bipartite graph $G = (L, R, E)$, we assume, w.l.o.g., that $L = R = \{0,1\}^{n/2}$ for some $n \in \mathbb{N}$ and represent it by a function $g : \{0,1\}^n \to \{0,1\}$, where $g(u \circ v) = 1$ for $u \in L, v \in R$ if and only if $(u, v) \in E$. Now we consider the partite function f of g and consider a SCSS for f creating shares $\mathsf{sh}_1, \ldots, \mathsf{sh}_{2n}$. We construct a SCSS for the forbidden graph G, where the share sh_u of vertex $u = (u_1, \ldots, u_{n/2}) \in L$ is $(\mathsf{sh}_{2-u_1}, \ldots, \mathsf{sh}_{n-u_{n/2}})$ and the share sh_v of vertex $v = (v_1, \ldots, v_{n/2}) \in V$ is $(\mathsf{sh}_{n/2+2-v_1}, \ldots, \mathsf{sh}_{2n-v_{n/2}})$. By the encoding of a function to a partite function, we obtain that a pair u, v can reconstruct the secret from $\mathsf{sh}_u, \mathsf{sh}_v$ if and only if $(u, v) \in E$ and the security holds whenever $(u, v) \notin E$. By plugging in Theorem 5.2 we derive the following theorem.

Theorem 5.5 (Theorem 1.8 restated). *Assume that there is a OWF with sub-exponential security. Then, for every graph $G = (V = \{0,1\}^{n/2}, E)$, represented by an adjacency matrix of size $N = 2^n$, there exists a SCSS for the forbidden graph G with share size $\operatorname{poly}(n,\lambda) = \operatorname{poly}(\log|V|, \lambda)$ that is 2^λ-secure. The sharing and reconstruction algorithms of the SCSS run in time $\operatorname{poly}(N, \lambda)$.*

The best known secret-sharing scheme with information-theoretic security for forbidden bipartite graphs (and arbitrary graphs) has share size $2^{\tilde{O}(\sqrt{\log n})}$ [LVW17]. Our SCSS is much more efficient.

6 SCSS for Monotone DNF and the Csirmaz Function from OWFs

Ito et al. [ISN87] showed that monotone DNF formulas can be realized by an information-theoretic secret-sharing scheme with a share size of $O(mn)$, where m is the number of terms and n is the number of variables. In the computational setting, one can generically reduce the share size to λ at the expense of increasing the public information to $O(mn)$ by using the transformation of Remark 2.8. In this section, we show that one can do better and reduce the public information size to only $O(m)$ bits. Moreover, if every term in the monotone formula contains many variables, e.g., $n/10$ variables, then, using ideas of Krawczyk [Kra94], we can construct a SCSS with share size $\lambda + O(m/n)$ without public information. Based on these ideas, we further show that computational secret-sharing schemes are provably more efficient than information-theoretic secret-sharing schemes. Specifically, we present a monotone function over n inputs which requires total share size $\Omega(n^2/\log n)$ in any information-theoretic secret-sharing scheme, as proved by Csirmaz [Csi97, Csi96], but admits a computational secret-sharing scheme whose total share size is $O(n\lambda)$. All the constructions in this section are based solely on the existence of one-way functions.

6.1 SCSS for Monotone DNF Formulas

We construct a SCSS for monotone DNF formulas by compressing the shares in the inormation-theoretic secret-sharing scheme of Ito et al. [ISN87]. Let φ be a monotone DNF formula over the variables $y_1, \ldots, y_n$. In a simple variant of the scheme of [ISN87], for each term $y_{i_1} \wedge y_{i_2} \wedge \cdots \wedge y_{i_k}$ in φ the secret is shared independently, that is, the dealer chooses k independent uniformly-distributed bits $r_1, \ldots, r_k$; it gives r_j as part of the share sh_{i_j} and publishes $s \oplus r_1 \oplus \cdots \oplus r_k$ as part of the public information. Thus, each share sh_i contains random bits, one bit per term containing y_i, and the public information contains m bits that are appropriately computed from the secret and shares of the parties. In this scheme each random bit is given to one party.

To compress the share of a party, its share will be a seed a_i of a pseudorandom generator G. Each seed a_i is expanded to the number of terms that contain y_i, and these bits are used to construct the public information as in the secret-sharing scheme [ISN87]. As one-way functions imply pseudorandom generators [HILL99], the above scheme can be based on one-way functions. As explained in Remark 4.12, the DNF scheme can be derived from the more general construction for monotone circuits (Theorem 4.10), and since there are no OR gates we do not need a pPRG and can use only standard PRGs.

Theorem 6.1 (Theorem 1.9 restated). *Assuming $t(\lambda)$-secure one-way functions, there exists a $\mathrm{poly}(t(\lambda))$-secure SCSS for monotone DNF formulas in which the share size is $O(\lambda)$ – the security parameter – and the public information size is m – the number of terms in the formula.*

Proof. Given a monotone DNF formula φ over n variables $y_1, \ldots, y_n$ and m terms let us denote by $T_i \subseteq [n]$ be the set of indices $j \in [n]$ such that y_j participates in the i^{th} term. For each party i, we set the i^{th} share sh_i to be a random seed $a_i \in_U \{0,1\}^\lambda$ and stretch a_i to a pseudorandom m-bit string r_i via an output-variable PRG $G(\cdot, 1^m) : \{0,1\}^\lambda \to \{0,1\}^m$ (such a PRG can be constructed from any one-way function). For each term $j \in [m]$ we publish the bit $s \oplus \bigoplus_{i \in T_j} r_i[j]$, where $r_i[j]$ is the j^{th} bit of the string r_i and s is the secret.

The complexity and correctness are easily verified. Let us prove security. Let $\mathcal{A}$ be any non-uniform $t(\lambda)$-time adversary. We will prove that its probability of winning the game of Definition 2.5 is at most $1/2 + \varepsilon(\lambda)$ for some negligible function $\varepsilon(\lambda)$. Fix a security parameter $\lambda \in \mathbb{N}$, set $t = t(\lambda)$, and let φ and $x \in \{0,1\}^n$ such that $\varphi(x) = 0$ be the DNF formula and input selected by the adversary in the first step of the game. Observe that $n, m \leq t$ since the adversary specifies the formula φ and the input x. Also, as always, we assume that $\lambda \leq t$. We define $n+1$ hybrids $H_0, \ldots, H_n$. The hybrid H_ℓ, where $0 \leq \ell \leq n$, is similar to the game defined in Definition 2.5 for the above secret-sharing scheme, except that for $i \leq \ell$ if $i \notin S_x$, the string r_i is sampled uniformly at random. Let p_ℓ be the probability that the adversary wins in the hybrid H_ℓ. Notice that the hybrid H_0 is the original game of the CSS and in the hybrid H_n all shares of the parties not in S_x are generated using truly random bits. By the information-theoretic security of the secret-sharing of [ISN87], the probability that the adversary wins in the hybrid H_n is $p_n = 1/2$ (since for every term the adversary misses at least one truly random bit of a party p_i such that $i \notin S_x$ and y_i is a variable in the term). We next argue that for every $1 \leq \ell \leq n$ there exists a negligible function ε_ℓ such that $|p_\ell - p_{\ell-1}| \leq \varepsilon_\ell$ by the security of the PRG G. If $x_\ell = 1$ the hybrids $H_{\ell-1}$ and H_ℓ are identical. Otherwise, consider an adversary for the PRG that is given a string r that is either random or pseudorandom and executes game defined in the hybrid H_ℓ except that it uses r as the random string r_ℓ of the ℓ^{th} party. Note that if r is a random string, then this is exactly the hybrid H_ℓ and if r is an output of the PRG, then this is exactly the hybrid $H_{\ell-1}$. Since the complexity of the PRG-adversary is $\mathrm{poly}(t, m, \lambda) = \mathrm{poly}(t)$, by the security of the PRG, there exists a negligible function ε_ℓ such that $|p_\ell - p_{\ell-1}| \leq \varepsilon_\ell(t)$. To conclude $p_0 = p_n + \sum_{\ell=1}^{n}(p_{\ell-1} - p_\ell) \leq p_n + \sum_{\ell=1}^{n} |p_\ell - p_{\ell-1}| \leq 1/2 + \sum_{\ell=1}^{n} \varepsilon_\ell$, and since the sum of $n = \mathrm{poly}(t)$ negligible functions is negligible, this proves the claim. □

We say that a monotone DNF formula over n variables is c-heavy if each term in the formula contains at least cn variables. Using ideas of [Kra94], the public information can be eliminated for heavy monotone DNF formulas, i.e., the public information can be efficiently dispersed between the parties by using any good erasure code (such as a Reed-Solomon code).

Lemma 6.2. *Let $0 < c < 1$ be a constant. Assuming one-way functions, there exists a SCSS for c-heavy monotone DNF formulas over n variables with $m > cn \log n$ terms in which the share size is $\lambda + m/cn$ and there is no public information.*

Proof. Let φ be a c-heavy monotone DNF formula with m terms. We execute the SCSS for the monotone DNF for φ, generating shares of size λ and public information of size m. We consider the public information as a string of length cn over an alphabet of $\{0,1\}^{m/cn}$ symbols, encode the string using a Reed-Solomon code to a code word of length n, and append the i^{th} symbol in this code-word to the share sh_i. Since $m \geq cn \log n$, the size of the alphabet is at least n and an appropriate Reed-Solomon code exists. As each term contains at least cn variables, each satisfying assignment x of φ is of weight at least cn, hence the shares of S_x contain at least cn symbols of the code-word, hence the shares of S_x can be used to decode the original m bits of the public information and reconstructs s. Security holds since the adversary's view in the current scheme can be simulated given its view in the previous (non-optimized) scheme. □

6.2 SCSS for the Csirmaz Function

In this section we show that computational secret-sharing schemes are provably more efficient than information-theoretic secret-sharing schemes. For this separation we use the best known lower bound for information-theoretic secret-sharing scheme of [Csi97, Csi96]. We first consider a simple function for which Csirmaz [Csi97] proved that there exists at least one party whose share size is $\Omega(n/\log n)$.

Definition 6.3 (The simple Csirmaz function C_n [Csi97])**.** *For every $n \in \mathbb{N}$, let k be the largest integer such that $2^k \le n$ (i.e., $k = \lfloor \log n \rfloor$). The function $C_n : \{0,1\}^{n+k} \to \{0,1\}$ is parameterized by some non-increasing ordering $(\mathsf{st}_1, \ldots, \mathsf{st}_{2^k})$ of all strings of length k. Here non-increasing means that*

$$\text{for every } i < i', \qquad \text{it holds that } \mathsf{st}_i \not\le \mathsf{st}_{i'}. \tag{7}$$

For example, we order the strings according to their weight, starting with $\mathsf{st}_1 = 1^k$, then strings of weight $k-1$, and so on, ending with $\mathsf{st}_{2^k} = 0^k$. The function $C_n : \{0,1\}^{n+k} \to \{0,1\}$ is the monotone function whose minterms are $1^i \circ 0^{n-i} \circ \mathsf{st}_i$ for $i = 1, \ldots, 2^k$, that is, the i^{th} minterm contains i ones, concatenated with $n-i$ zeros, concatenated by st_i.

In C_n the variables are partitioned to a small set of size $\log n$ and a big set of size n; Csirmaz [Csi97] proved that in any information-theoretic secret-sharing scheme realizing C_n there exists an index i from the small set such that $|\mathsf{sh}_i| = \Omega(n/\log n)$. The function C_n can be computed by a monotone DNF formula with $2^k \le n$ terms. Thus, we can construct a SCSS for C_n with share size λ and public information size n (one bit per term). Our goal is a scheme without public information. For C_n this is simple, we append the public bit of the term corresponding to $1^i \circ 0^{n-i} \circ \mathsf{st}_i$ to the share sh_i. That is, each share $\mathsf{sh}_1, \ldots, \mathsf{sh}_n$ gets one "public" bit, and its size is $\lambda + 1$.[11]

Theorem 6.4. *Assuming one-way functions, the above scheme is a secure computational secret-sharing scheme for $(C_n)_{n \in \mathbb{N}}$ in which the size of each share is at most $\lambda + 1$.*

Theorem 6.4 already provides a separation of $\Omega(n/\log n)$ vs. $O(\lambda)$ with respect to the max-share size. Next, we show how to derive a stronger separation with respect to the *total* share size. For this, we will consider the following variant of the function of [Csi96].

Definition 6.5 (A variant of the full Csirmaz function [Csi96])**.** *For every $n \in \mathbb{N}$, define a monotone function $C'_n : \{0,1\}^{2n} \to \{0,1\}$ as follows: Let k be the largest integer such that $2^k \le n$ and $L = \lfloor n/k \rfloor$, and define*

$$C'_n(x_1, \ldots, x_{2n-k\cdot L}, y_{1,1}, \ldots, y_{1,L}, \ldots, y_{k,1}, \ldots, y_{k,L}) = C_n(x_1, \ldots, x_n, \bigvee_{\ell=1}^{L} y_{1,\ell}, \ldots, \bigvee_{\ell=1}^{L} y_{k,\ell}).$$

In C'_n there is one big set of variables as in C_n and $O(n/\log n)$ copies of the small set of C_n.

[11]Since C_n is a single function, we assume that C_n is represented by 1^n. This is aligned with the assumption that representation is always at least as large as the input size and it also means that n can be taken to be an arbitrary polynomial in the security parameter since 1^n is specified by a $\mathrm{poly}(\lambda)$-time adversary.

Theorem 6.6. *Assuming that one-way functions, there exists a SCSS for $(C'_n)_{n\in\mathbb{N}}$ with share size $O(\lambda)$ and total share size $O(n\lambda)$. In contrast, in every information-theoretic secret-sharing scheme for C'_n the total share size is $\Omega(n^2/\log n)$.*

Theorem 1.10 follows by taking λ to n for a constant > 0. (Recall that n can be an arbitrary polynomial in λ.)

Proof. We employ an output-variable PRG whose existence follows from the existence of OWFs. By Theorem 6.4, there exists a computational secret-sharing scheme realizing C_n with share size $O(\lambda)$ and total share size $O(n\lambda)$. In C'_n there are L copies of the j-party for $n+1 \leq j \leq n+k$. To construct a SCSS for C'_n, we execute the SCSS for C_n and give the share of the j^{th} party, where $n+1 \leq j \leq n+k$, to its L copies. Thus, each share in the SCSS for C'_n has size $O(\lambda)$.

To prove the lower bound on the share size of information-theoretic secret-sharing schemes realizing C'_n, we use a result of [Csi97]: In every information-theoretic secret-sharing schemes realizing C_n the sum of the sizes of $\mathsf{sh}_{n+1},\dots,\mathsf{sh}_{n+k}$ is $\Omega(n)$. For every $j \in [k]$, the function C'_n projected to the variables $x_1,\dots,x_n,y_{1,j},\dots,y_{k,j}$ (i.e., all other variables are fixed to zero) is isomorphic to C_n, thus, in every information-theoretic secret-sharing schemes realizing C'_n the sum of the sizes of $\mathsf{sh}'_{1,j},\dots,\mathsf{sh}'_{k,j}$ is $\Omega(n)$ and the sum of the sizes of $(\mathsf{sh}_{i,j})_{1\leq i\leq k, 1\leq j\leq L}$ is $\Omega(Ln) = \Omega(n^2/\log n)$. □

Acknowledgements. BA was supported by the ISF grant 2805/21. AB is supported by ERC grant 742754 (project NTSC) and ISF grant 391/21. YI was supported by ERC grant 742754 (project NTSC), BSF grant 2018393, ISF grant 2774/20, and a Google Faculty Research Award. EK was supported by BSF grant 2018393 and ISF grant 2774/20. VV was supported by DARPA under Agreement No. HR00112020023, NSF CNS-2154149 and a Thornton Family Faculty Research Innovation Fellowship from MIT. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. VV thanks Hoeteck Wee for quick and comprehensive answers to his questions on broadcast encryption.

References

[ABF+19] Benny Applebaum, Amos Beimel, Oriol Farràs, Oded Nir, and Naty Peter. Secret-sharing schemes for general and uniform access structures. In *EUROCRYPT 2019*, volume 11478 of *LNCS*, pages 441–471. Springer, 2019. 1, 7, 38

[ABN+22] Benny Applebaum, Amos Beimel, Oded Nir, Naty Peter, and Toniann Pitassi. Secret sharing, slice formulas, and monotone real circuits. In *13th ITCS*, volume 215 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 1

[ABNP20] Benny Applebaum, Amos Beimel, Oded Nir, and Naty Peter. Better secret sharing via robust conditional disclosure of secrets. In *52th STOC*, pages 280–293. ACM, 2020. 1, 7, 9, 10, 38

[AIKW15] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate, or how to compress garbled circuit keys. *SIAM J. on Computing*, 44(2):433–466, 2015. 4, 18, 19, 25

[AIR01] Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 118–134. Springer, 2001. 7

[AN21] Benny Applebaum and Oded Nir. Upslices, downslices, and secret-sharing with complexity of 1.5^n. In *CRYPTO 2021*, volume 12827, pages 627–655. Springer, 2021. 1, 5, 7, 38

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In *EUROCRYPT 2022*, volume 13275 of *LNCS*, pages 790–820. Springer, 2022. 4

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, 2014. 7

[BCG$^+$18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018. 11

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019*, volume 11694 of *LNCS*, pages 489–518. Springer, 2019. 4

[BD91] Ernest F. Brickell and Daniel M. Davenport. On the classification of ideal secret sharing schemes. *J. of Cryptology*, 4(73):123–134, 1991. 6

[BDDV97] Carlo Blundo, Alfredo De Santis, Roberto De Simone, and Ugo Vaccaro. Tight bounds on the information rate of secret sharing schemes. *Designs, Codes and Cryptography*, 11(2):107–122, 1997. 6

[Bei23] Amos Beimel. Lower-bounds for secret-sharing schemes for k-hypergraphs. *IACR Cryptol. ePrint Arch.*, page 289, 2023. 9, 10

[BF20] Amos Beimel and Oriol Farràs. The share size of secret-sharing schemes for almost all access structures and graphs. In *TCC 2020*, volume 12552 of *LNCS*, pages 499–529. Springer, 2020. 6

[BFM16] Amos Beimel, Oriol Farràs, and Yuval Mintz. Secret-sharing schemes for very dense graphs. *J. of Cryptology*, 29(2):336–362, 2016. 6, 30

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, 2014. 9, 10, 53

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001. 23

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, 2014. 10, 23

[BGW99] László Babai, Anna Gál, and Avi Wigderson. Superpolynomial lower bounds for monotone span programs. *Combinatorica*, 19(3):301–319, 1999. 1

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO 2005*, volume 3621 of *LNCS*, pages 258–275. Springer, 2005. 8, 10

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, 2008. 26

[BI05] Amos Beimel and Yuval Ishai. On the power of nonlinear secret-sharing. *SIAM J. on Discrete Mathematics*, 19(1):258–280, 2005. 7

[BIKK14] Amos Beimel, Yuval Ishai, Ranjit Kumaresan, and Eyal Kushilevitz. On the cryptographic complexity of the worst functions. In *TCC 2014*, volume 8349 of *LNCS*, pages 317–342. Springer, 2014. 41

[BL90] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *CRYPTO 1988*, volume 403 of *LNCS*, pages 27–35. Springer, 1990. 1, 11

[Bla79] G. Robert Blakley. Safeguarding cryptographic keys. In *Proc. of the 1979 AFIPS National Computer Conference*, volume 48 of *AFIPS Conference proceedings*, pages 313–317. AFIPS Press, 1979. 1

[BR07] Mihir Bellare and Phillip Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *CCS 2017*, pages 172–184. ACM, 2007. 11

[BS92] Ernest F. Brickell and Douglas R. Stinson. Some improved bounds on the information rate of perfect secret sharing schemes. *J. of Cryptology*, 5(3):153–166, 1992. 6

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In *TCC 2017*, volume 10677 of *LNCS*, pages 264–302. Springer, 2017. 53

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC 2015*, volume 9015 of *LNCS*, pages 1–30. Springer, 2015. 53

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013*, volume 8270 of *LNCS*, pages 280–300. Springer, 2013. 10, 23

[Cac95] Christian Cachin. On-line secret sharing. In *Cryptography and Coding, 5th IMA Conference*, volume 1025 of *LNCS*, pages 190–198. Springer, 1995. 11

[CDG$^+$17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In *CRYPTO 2017*, volume 10402 of *LNCS*, pages 33–65. Springer, 2017. 8

[CMM16] Melissa Chase, Mary Maller, and Sarah Meiklejohn. Déjà Q all over again: Tighter and broader reductions of q-type assumptions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 655–681, 2016. 10

[CSGV93] Renato M. Capocelli, Alfredo De Santis, Luisa Gargano, and Ugo Vaccaro. On the size of shares for secret sharing schemes. *J. of Cryptology*, 6(3):157–168, 1993. 6

[Csi96] László Csirmaz. The dealer's random bits in perfect secret sharing schemes. *Studia Sci. Math. Hungar.*, 32(3–4):429–437, 1996. 1, 5, 8, 42, 44

[Csi97] László Csirmaz. The size of a share must be large. *J. of Cryptology*, 10(4):223–231, 1997. 1, 8, 42, 44, 45

[Csi05] László Csirmaz. Secret sharing schemes on graphs. Technical Report 2005/059, Cryptology ePrint Archive, 2005. `eprint.iacr.org/`. 6

[Csi09] László Csirmaz. An impossibility result on graph secret sharing. *Designs, Codes and Cryptography*, 53(3):195–209, 2009. 6

[Csi15] László Csirmaz. Secret sharing on the d-dimensional cube. *Designs, Codes and Cryptography*, 74(3):719–729, 2015. 6

[CT13] László Csirmaz and Gábor Tardos. Optimal information rate of secret sharing schemes on trees. *IEEE Trans. Inf. Theory*, 59(4):2527–2530, 2013. 6

[DGI$^+$19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In *CRYPTO 2019*, volume 11694 of *LNCS*, pages 3–32. Springer, 2019. 8

[EP97] Paul Erdös and Lászlo Pyber. Covering a graph by complete bipartite graphs. *Discrete Mathematics*, 170(1–3):249–251, 1997. 5, 6

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *19th STOC*, pages 427–437, 1987. 11

[FKMP18] Oriol Farràs, Tarik Kaced, Sebastià Martín, and Carles Padró. Improving the linear programming technique in the search for lower bounds in secret sharing. In *EUROCRYPT 2018*, LNCS, pages 597–621. Springer-Verlag, 2018. 6

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO 1993*, volume 773 of *LNCS*, pages 480–491. Springer, 1994. 8

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 191–219. Springer, 2015. 7, 38

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013. 23

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. of the ACM*, 33(4):792–807, 1986. 12

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. of Computer and System Sciences*, 60(3):592–629, 2000. 7, 38, 41

[GKW15] Romain Gay, Iordanis Kerenidis, and Hoeteck Wee. Communication complexity of conditional disclosure of secrets and attribute-based encryption. In *CRYPTO 2015*, volume 9216 of *LNCS*, pages 485–502. Springer, 2015. 7

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st STOC*, pages 25–32. ACM, 1989. 20

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001. 12

[GPSW06] V. Goyal, O. Pandey, A. Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006. 8

[HIJ$^+$16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *ITCS 2016*, pages 157–168. ACM, 2016. 4

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Construction of a pseudo-random generator from any one-way function. *SIAM J. on Computing*, 28(4):1364–1396, 1999. 12, 35, 39, 42

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS 2015*, pages 163–172. ACM, 2015. 4, 23

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *TCC 2015*, volume 9015 of *LNCS*, pages 668–697. Springer, 2015. 11

[ISN87] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *Globecom 1987*, pages 99–102, 1987. Journal version: Multiple assignment scheme for sharing secret. J. of Cryptology 6(1), 15-20, (1993). 1, 5, 8, 32, 42, 43

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *41st ICALP*, pages 650–662, 2014. 7, 38

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In *EUROCRYPT 2022*, volume 13275 of *LNCS*, pages 670–699. Springer, 2022. 4

[KGH83] Ehud D. Karnin, Jonathan W. Greene, and Martin E. Hellman. On secret sharing systems. *IEEE Trans. on Information Theory*, 29(1):35–41, 1983. 11

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *45th STOC*, pages 419–428. ACM, 2015. 11

[KNY17] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. *J. Cryptol.*, 30(2):444–469, 2017. 2, 11

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 669–684. ACM, 2013. 10, 23

[Kra94] Hugo Krawczyk. Secret sharing made short. In *CRYPTO 1993*, volume 773 of *LNCS*, pages 136–146. Springer, 1994. 2, 11, 14, 42, 43

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proc. of the 8th IEEE Structure in Complexity Theory*, pages 102–111, 1993. 1

[LS20] Kasper Green Larsen and Mark Simkin. Secret sharing lower bound: Either reconstruction is hard or shares are long. In *SCN 2020*, volume 12238 of *LNCS*, pages 566–578. Springer, 2020. 33

[LV18] Tianren Liu and Vinod Vaikuntanathan. Breaking the circuit-size barrier in secret sharing. In *48th STOC*, pages 699–708, 2018. 1, 7, 38

[LVW17] Tianren Liu, Vinod Vaikuntanathan, and Hoteck Wee. Conditional disclosure of secrets via non-linear reconstruction. In *CRYPTO 2017*, volume 10401 of *LNCS*, pages 758–790. Springer, 2017. 5, 38, 41

[LVW18] Tianren Liu, Vinod Vaikuntanathan, and Hoteck Wee. Towards breaking the exponential barrier for general secret sharing. In *EUROCRYPT 2018*, LNCS, pages 758–790. Springer, 2018. 1, 5, 7

[ODK$^+$17] Vanga Odelu, Ashok Kumar Das, Muhammad Khurram Khan, Kim-Kwang Raymond Choo, and Minho Jo. Expressive CP-ABE scheme for mobile devices in iot satisfying constant-size keys and ciphertexts. *IEEE Access*, 5:3273–3283, 2017. 18

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 121–145. Springer, 2015. 4

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 129–140. Springer, 1991. 11

[Pet20] Naty Peter. *Secret-Sharing Schemes and Conditional Disclosure of Secrets Protocols*. PhD thesis, Ben-Gurion University of the Negev, 2020. https://primo.bgu.ac.il/permalink/972BGU_INST/23v028/alma9926575584104361. 9, 10

[PR17] Toniann Pitassi and Robert Robere. Strongly exponential lower bounds for monotone computation. In *47th STOC*, pages 1246–1255, 2017. 1

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *59th FOCS*, pages 859–870. IEEE Computer Society, 2018. 8

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979. 1

[Sha81] Adi Shamir. On the generation of cryptographically strong pseudo-random sequences. In *8th ICALP*, volume 115 of *LNCS*, pages 544–550. Springer, 1981. 22

[SS97] Hung-Min Sun and Shiuh-Pyng Shieh. Secret sharing in graph-based prohibited structures. In *INFOCOM 1997*, pages 718–724, 1997. 7, 41

[SW05] A. Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT 2005*, pages 457–473. Springer, 2005. 8

[vD95] Marten van Dijk. On the information rate of perfect secret sharing schemes. *Designs, Codes and Cryptography*, 6(2):143–169, 1995. 6

[VNS$^+$03] Vinod Vaikuntanathan, Arvind Narayanan, Kannan Srinathan, C. Pandu Rangan, and Kwangjo Kim. On the power of computational secret sharing. In *Indocrypt 2003*, volume 2904 of *LNCS*, pages 162–176. Springer, 2003. 6, 11, 33

[VV15] Vinod Vaikuntanathan and Prashant Nalini Vasudevan. Secret sharing and statistical zero knowledge. In *ASIACRYPT 2015*, pages 656–680, 2015. 7

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, 2014. 7

[Wee16] Hoeteck Wee. Déjà Q: encore! un petit IBE. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 237–258. Springer, 2016. 10

[Yao89] Andrew Chi-Chih Yao. Unpublished manuscript, 1989. Presented at Oberwolfach and DIMACS workshops. 2, 6, 11, 33

A Projective PRG from LWE

We describe a projective PRG based on the LWE assumption. In contrast to the DDH-based construction, we show how to make the size of the (non-reusable) public parameters linear in m (as opposed to quadratic).

Assumption A.1 (The learning with errors assumption). *Let $n = n_\lambda \in \mathbb{N}, q = q_\lambda \in \mathbb{N}$ be positive integers and let $\chi = \chi_\lambda$ be a probability distribution over $\mathbb{Z}$. Let $\mathcal{O}_{\mathbf{s}}$ be an oracle that, on every invocation, samples a uniformly random $\mathbf{a} \in \mathbb{Z}_q^n$, $e \leftarrow \chi$ and returns the pair $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e \bmod q)$, and let $\mathcal{O}_{rand}$ be an oracle that returns $(\mathbf{a}, b)$, where $\mathbf{a}$ and $b \in \mathbb{Z}_q$ are chosen uniformly at random from the corresponding domains. The (t, ε)-(decisional)-learning with errors (LWE) assumption with respect to these parameters (n, q, χ) requires that for every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$,*

$$\Pr_{\mathbf{s}\leftarrow\mathbb{Z}_q^n}\left[\mathcal{A}^{\mathcal{O}_{\mathbf{s}}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_{rand}}(1^\lambda) = 1\right] \le \varepsilon(\lambda),$$

where the probabilities are over the coin-tosses of both the oracle and the adversary as well as the random choice of the LWE secret $\mathbf{s}$.

Theorem A.2. *Under the LWE assumption, there exists a robust pPRG whose projective keys are strongly succinct, i.e., of length $O(\lambda)$. The public parameter is of length $m \cdot \mathsf{poly}(\lambda)$, where m is the output length.*

Our starting point is a construction from LWE that is very similar to the one from DDH. That is, the public parameters consists of an ℓ-by-ℓ matrix $\mathbf{M} \in \mathbb{Z}_q^{\ell \times \ell}$ whose $(i,j)^{th}$ entry is $\langle \mathbf{a}_i, \mathbf{s}_j \rangle + e_{ij} \bmod q$ if $i \neq j$ and $\langle \mathbf{a}_i, \mathbf{s}_j \rangle + e_{ij} + \Delta y_i \bmod q$, where $\mathbf{a}_i \leftarrow \mathbb{Z}_q^n$ and $\mathbf{s}_j \leftarrow \mathbb{Z}_q^n$ are uniformly random, $e_{ij} \leftarrow \chi$ is a small error, $\Delta = \lfloor q/p \rceil$ for some plaintext modulus $p < q$, and $y_i \leftarrow \mathbb{Z}_p$. The vectors $\{\mathbf{a}_i\}_{i \in [\ell]}$ are also published as part of the public parameters while the vectors $\{\mathbf{s}_j\}_{j \in [\ell]}$ are kept secret. Similar to the DDH construction, a projective key for a subset $T \subseteq [\ell]$ is

$$\mathsf{msk}\{T\} := \sum_{j \in T} \mathbf{s}_j \ .$$

Given $\mathsf{msk}\{T\}$, one can recover the set of outputs $\{y_j\}_{j \in T}$ just as in the DDH-based construction.

The key new observation that enables shrinking the public parameters is that it is possible to generate the vectors $\mathbf{a}_i$ in a special "pseudorandom" way, that additionally enables us to compress each column of the matrix $\mathbf{M}$ to size polynomial in $\log \ell$ and λ, all the while maintaining security. In particular, we will choose a set of *random matrices* $\{\mathbf{A}'_i\}_{i \in [\ell']}$, where $\ell' \approx \log \ell$, and define the vectors $\{\mathbf{a}_i\}_{i \in [\ell]}$ as the function of $\{\mathbf{A}'_i\}_{i \in [\ell']}$. The machinery necessary to accomplish this and to compress each column of $\mathbf{M}$ comes from the literature on constrained pseudorandom functions (PRFs), in particular constructions of these objects from LWE [BGG+14, BV15]. We will use the following lemma from [BTVW17].

Lemma A.3. *There are algorithms* ExpandA, CompressC, ExpandC *that work as follows:*

- *Given a sequence of matrices* $\mathbf{A}'_1, \ldots, \mathbf{A}'_{\ell'} \in \mathbb{Z}_q^{n \times m}$ *(where* $m = cn \log q$ *for some constant* $c > 1$*, and* $\ell' = \mathsf{poly}(\log \ell)$*),* ExpandA *outputs a sequence of* ℓ *vectors* $\mathbf{a}_1, \ldots, \mathbf{a}_\ell$.
- *Given* $\mathbf{A}'_1, \ldots, \mathbf{A}'_{\ell'}$*, an LWE secret* $\mathbf{s} \in \mathbb{Z}_q^n$ *and a function* $f : \{0,1\}^{\log \ell} \to \{0,1\}$*,* CompressC *outputs a program* $\Pi_{\{\mathbf{A}'_i\}, f, \mathbf{s}}$ *whose size is polynomial in* $\log \ell, n, \log q$ *and the description size of the circuit* f.
- *Given the program* $\Pi_{\{\mathbf{A}'_i\}, f, \mathbf{s}}$*,* ExpandC *outputs* $b_j \approx \langle \mathbf{a}_j, s \rangle$ *for all* j *such that* $f(j) = 1$.

The security property says the converse, that is, given $\Pi_{\{\mathbf{A}'_i\}, f, \mathbf{s}}$*, the set of all*

$$\{\langle \mathbf{a}_j, \mathbf{s} \rangle + e_j : f(j) = 0\}$$

is jointly pseudorandom (where $e_j \leftarrow \chi$*).*

The construction of the pPRG proceeds as follows.

- On input 1^λ and ℓ, choose parameters $n = n(\lambda)$, $q = q(\lambda)$, and $\chi = \chi(\lambda)$ to achieve security against adversaries running in time polynomial in λ and ℓ.
- The parameter generation algorithm chooses uniformly random matrices $\mathbf{A}'_1, \ldots, \mathbf{A}'_{\ell'}$, uniformly random LWE secrets $\mathbf{s}_1, \ldots, \mathbf{s}_\ell$, uniformly random errors $e_j \leftarrow \chi$, uniformly random numbers $y_j \leftarrow \mathbb{Z}_p$, and outputs ℓ programs, together with auxiliary information as follows:

  $$\Pi^{(j)}_{\{\mathbf{A}'_i\}, f_j, \mathbf{s}} \leftarrow \mathsf{CompressC}(\mathbf{A}'_1, \ldots, \mathbf{A}'_{\ell'}, \mathbf{s}_j, f_j) \ \text{ and } \ c_j = \langle \mathbf{a}_j, \mathbf{s}_j \rangle + e_j + \Delta y_j \bmod q,$$

  where $f_j(i) = 1$ if and only if $i = j$.

  Note that the public parameters depend on the PRG seed, and are non-reusable.

- The projective key for a subset $T \subseteq [n]$ is $\sum_{j \in T} \mathbf{s}_j$.
- If $i \in T$, the i^{th} output of the PRG can be recovered by first computing the j^{th} columns for all $j \in T$ using the public parameters by running

  $$\mathbf{m}_{-j} \leftarrow \Pi^{(j)}_{\{\mathbf{A}'_i\}, f_j, \mathbf{s}}, \text{ where } \mathbf{m}_{-j} \in \mathbb{Z}_q^{\ell-1}$$

  and inserting c_j in the (j, j) position to get the entire j^{th} column $\mathbf{m}_j$.

  Summing up all the $\mathbf{m}_j$ for $j \in T$ results in $\approx \langle \mathbf{a}_j, \sum_{j \in T} \mathbf{s}_j \rangle + \Delta y_j$ for all $j \in T$. Using the projective key, one can now recover y_j.

The security proof goes exactly analogously to the proof of the DDH construction, using in addition the security property of the algorithms $(\mathsf{ExpandA}, \mathsf{CompressC}, \mathsf{ExpandC})$. We defer the details to the full version.

By applying similar balancing tricks as in Corollary 3.15, we obtain the following corollary.

Corollary A.4. *Under the LWE assumption, for every $\delta \in [0, 1]$ there exists a robust pPRG whose projective keys have length $m^\delta \cdot \text{poly}(\lambda)$ and the public parameter has length $m \cdot \text{poly}(\lambda)$, where the evaluation algorithm needs only $m^{1-\delta} \cdot \text{poly}(\lambda)$ bits from the public parameter to evaluate a single bit c_i.*