

Threshold BARGs with Statistical Soundness: Weighted Threshold Encryption without Obfuscation

Shafik Nassar
UT Austin
shafik@cs.utexas.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

Non-interactive batch arguments (BARGs) for NP enable proving the validity of k NP statements with a proof whose size is independent of k . Threshold BARGs are a natural generalization that enables proving that at least t out of k statements are valid. Unfortunately, existing threshold BARG constructions have the drawback that the size of the common reference string (CRS) grows polynomially with k . This is the case even when the statements have a short description (e.g., when considering index languages). Moreover, other than constructions based on indistinguishability obfuscation (*iO*), prior constructions could only achieve *computational* soundness.

In this work, we construct the first threshold BARG with *statistical* soundness for an expressive class of trapdoor threshold languages from either circular-secure fully homomorphic encryption or from witness encryption in combination with the learning with errors assumption. Moreover and importantly for our applications, the size of the CRS only scales polylogarithmically with k .

Next, we combine our threshold BARG with a new technique for sampling pseudorandom permutations in order to build *weighted* threshold signatures with a statistical unforgeability guarantee. Then, in combination with witness encryption, we obtain a *weighted* threshold encryption scheme that supports λ -bit weights. Decryption is entirely non-interactive in our setting. Previously, such a scheme was only known from *iO*. If we additionally work in the random oracle model, then we obtain a weighted threshold encryption scheme that supports arbitrary weights with *silent* setup. In the silent setup model, users choose their own public/secret keys and no longer need to rely on a central authority to generate and issue decryption keys.

Contents

1	Introduction	3
1.1	This Work	4
2	Technical Overview	6
2.1	Puncturable Threshold Signatures and Weighted Threshold Encryption	10
3	Preliminaries	14
3.1	Cryptographic Primitives	15
4	Building Blocks	18
4.1	Dynamic Somewhere-Extractable Hash	18
4.2	Statistical Functional Non-Interactive Batch Arguments	20
5	BARG for Trapdoor Threshold Languages	22
5.1	Construction	24
6	Pseudorandom Permutations with Efficient Queries	36
7	Puncturable Public-Coin Threshold Signatures	42
7.1	Our Construction	45
8	Puncturable Public-Coin Weighted Threshold Signatures	54
8.1	A Generic Compiler from Puncturable Public-Coin Threshold Signatures	56
9	Public-Coin Weighted Threshold Encryption	63
9.1	Our Construction	65
10	Silent Setup in the Random Oracle Model	71
A	Dynamic Somewhere-Extractable Hash Construction	75
A.1	Building Block	76
A.2	Construction	77
B	Statistical Functional BARGs	83
B.1	Building Blocks	84
B.2	Constructing BARGs	86
B.3	Security Proofs	91
B.4	Putting It Together	97

1 Introduction

A non-interactive batch argument for NP (BARG) [BHK17, KPY19] allows a prover to convince a verifier that a batch of k NP statements $x_1, \dots, x_k$ are true with a single proof whose length is independent of k . More recent works have extended BARGs to a richer class of monotone policies [BBK⁺23, BCJP24, NWW24, NWW25, DJWW25], where instead of proving that all k NP statements are true, the prover proves that there is a subset $S \subseteq [k]$ of statements $\{x_i\}_{i \in S}$ which are true and moreover, that the subset S satisfies a monotone policy P . For instance, the policy P could be a threshold policy which requires that at least t out of the k statements are true. Existing constructions of monotone-policy BARGs typically suffer from two main drawbacks:

- **Computational soundness:** First, batch arguments are *computationally* sound, which means a computationally-bounded prover cannot come up with a proof for a collection of statements $x_1, \dots, x_k$ where x_i is false for some $i \in [k]$ (or more generally, in the setting of monotone-policy BARGs, when there does not exist a set of true statements that satisfies the policy). Computational soundness does not rule out the possibility that there exists a proof on such a collection of statements; it only stipulates that such proofs are hard to find. While computational soundness suffices for many applications (cf. [CJJ21b, WW22, DGKV22, PP22, BCJP24, BDS25, HSY26, PP26]), certain applications may require the stronger notion of *statistical* soundness (i.e., batch *proofs*). Achieving statistical soundness for all of NP would have surprising complexity-theoretic implications [BKP⁺24], but as we show in this work, we can achieve statistical soundness for a meaningful subset of trapdoor relations.
- **Large CRS size:** Second, many existing constructions of monotone-policy BARGs [BBK⁺23, BCJP24, NWW24, NWW25] require a large common reference string (CRS) whose size scales *polynomially* with the number of instances k (and more broadly, with the width of the circuit computing the associated policy P). A natural goal is to obtain a monotone-policy BARG with a short CRS (i.e., size independent of k). Currently, this is known for the special case of vanilla BARGs (i.e., the case of a conjunction policy) [CJJ21b, DGKV22, PP22, CGJ⁺23, KLVW23] or by relying on indistinguishability obfuscation [DJWW25].

A natural question is whether we can construct monotone-policy BARGs that satisfy statistical soundness and have a short CRS (independent of the number of instances) for an interesting policy family and a non-trivial subset of NP.

A motivating application: weighted threshold encryption. Consider an application to threshold encryption [Des87, Fra89]. In this model, there are k different users $\mathcal{P}_1, \dots, \mathcal{P}_k$, and each user $i \in [k]$ holds a secret key sk_i . Anyone can encrypt a message msg with respect to a public encryption key ek and a threshold $t \leq k$. The size of the resulting ciphertext ct should be independent of k . To decrypt a ciphertext ct , any user $\mathcal{P}_i$ can locally decrypt ct with their secret key sk_i and recover a decryption hint dh_i . Given a collection of t decryption hints from different users, one can recover the message. Notably, decryption should be *non-interactive* where users just locally decrypt the ciphertext with their secret key. Security says that the ciphertext should computationally hide the message from any subset with fewer than t users. In a weighted threshold encryption scheme, we additionally associate a weight w_i with each user. Given decryption hints from any set of users $S \subseteq [k]$ where $\sum_{i \in S} w_i \geq t$, it should be possible to recover the message. The ciphertext should computationally hide the message against any set of users T where $\sum_{i \in T} w_i < t$.

The classic approach to build weighted threshold encryption is to start from a vanilla threshold encryption scheme and use virtualization. Here, a user with weight w_i would be treated as w_i virtual parties and receive w_i independent keys for the underlying threshold encryption scheme. Virtualization blows up the parameters of the scheme by the *magnitude* of the weights, and as such, the weighted scheme can only support polynomial-size weights (i.e., $w_i = \text{poly}(\lambda)$, where λ is the security parameter). Prior to this work, existing constructions of weighted threshold encryption were limited to polynomial-size weights [GJM⁺23, ABD⁺26, FPRT26, Pol26b, Pol26a], or if they could support exponential-size weights (i.e., λ -bit weights), this either required an *interactive* decryption procedure [GSW26] or the full power of indistinguishability obfuscation [DJWW25].

The work of [DJWW25] proposes a general template for designing threshold encryption by combining a threshold BARG with a witness encryption scheme. First, recall that in a witness encryption scheme [GGSW13], anyone can encrypt a message to an NP statement x . A user who knows a witness w for x can then recover the message. Security says that if x is false, then the ciphertext computationally hides the message. When x is true, witness encryption

provides no guarantees on hiding, even if it is computationally infeasible to find a valid witness w for x . A simple template for building threshold encryption from witness encryption is as follows:

- Each user $\mathcal{P}_i$ has a public key pk_i and a secret key sk_i . Let ek be a Merkle-style hash [Mer89] of $(pk_1, \dots, pk_k)$.
- To encrypt a message msg to the users, we encrypt with respect to the NP relation that takes the encryption key ek and a threshold BARG proof π as input and checks that π proves knowledge of at least t -out-of- k secret keys in the system. Here the threshold BARG relation treats the index i as the statement and the witness consists of a local opening of ek to pk_i as well as the secret key sk_i associated with pk_i . The basic template here does not support weights, but as we will show later (see Section 2), it is not difficult to do so.

Observe that to implement this template, we require two properties on the BARG: (1) the BARG verification algorithm must be succinct (independent of the number of users k) as otherwise, the ciphertext size scales linearly with k ; and (2) the BARG needs to be statistically sound for false statements, as otherwise, we *cannot* invoke security of the witness encryption.

Tension between succinctness and statistical soundness. It is well known that succinct *proofs* for NP are unlikely, even if we allow for interaction [BHZ87, GH98, GVW01, Wee05]. Even if we consider the batch setting where we allow the size of the proof for k statements to grow with the size of the NP relation, but not with k , such a scheme would have surprising complexity-theoretic implications [BKP⁺24]. A natural relaxation then is to consider a weaker class of trapdoor languages as done in a number of recent works [DJWW25, DJJ⁺25, DHJQ26, Nas26]. Trapdoor languages are languages where there is an algorithm that can efficiently identify or “flag” a subset of the false statements (given a trapdoor). Recently, the works of [DHJQ26, Nas26] showed how to construct dual-mode succinct non-interactive arguments, where the common reference string can be sampled in one of two computationally indistinguishable modes: normal mode and binding mode. The binding setup takes as input a trapdoor for the language, and guarantees that instances flagged by the trapdoor have no accepting proofs. However, even for trapdoor languages, building statistically-sound arguments remains challenging, and both works critically rely on indistinguishability obfuscation (*iO*) [BGI⁺01, GGH⁺13]. A natural question is whether we can achieve succinctness and statistical soundness without obfuscation (and for our application to threshold encryption, without needing a large CRS).

1.1 This Work

Our first contribution in this work is a new non-interactive *threshold* batch argument that achieves *statistical* soundness for a sub-class of index languages. Here, a batch argument for an index language [CJJ21b] is one where the statements just consist of the integers $\{1, \dots, k\} = [k]$. In more detail, let $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ be a Boolean circuit and let $t \in [0, k]$ be a threshold. We say the circuit satisfies the threshold if there exist at least t satisfiable indices $i \in [k]$; namely, that

$$|\{i \in [k] : \exists w \in \{0, 1\}^m \text{ such that } C(i, w) = 1\}| \geq t.$$

A non-interactive threshold batch argument is a succinct non-interactive argument system for the threshold language

$$\mathcal{L} = \{(C, t) : C \text{ satisfies the threshold } t\}.$$

Next, at a very high level, we say an algorithm $\text{Ext}: [k] \times [k] \rightarrow [0, k]$ is a trapdoor for the threshold language associated with C if it computes an upper bound on the number of *satisfiable* indices in any given interval: namely,

$$\text{Ext}(j_1, j_2) \geq |\{i \in [j_1, j_2] : \exists w \in \{0, 1\}^m \text{ such that } C(i, w) = 1\}|. \quad (1.1)$$

We refer to Section 2 for a more precise definition. Note that a single algorithm Ext can be a trapdoor for multiple distinct circuits. The threshold BARG we build provides statistical soundness on instances (C, t) whose circuit C has an efficiently-computable trapdoor Ext . More formally, a BARG for trapdoor threshold languages has a normal setup mode and a binding setup mode that takes as input a trapdoor Ext . The two modes are computationally indistinguishable. In binding mode, it is guaranteed that for any instance (C, t) with $\text{Ext}(1, k) < t$, no accepting proof

exists. This is a special case of so-called *sometimes statistically sound* succinct non-interactive arguments [Nas26]. Both the size of the common reference string and the size of the proof grow only polylogarithmically with k (and polynomially with the security parameter λ , the circuit size $|C|$, and the trapdoor size $|\text{Ext}|$). We give a construction of such an argument system from witness encryption and the learning with errors assumption with a sub-exponential modulus-to-noise ratio, or, alternatively, from any circular-secure fully homomorphic encryption alone:

Theorem 1.1 (BARG for Trapdoor Threshold Languages; Informal). *Assume the existence of witness encryption and the learning with errors (LWE) assumption with a sub-exponential modulus-to-noise ratio, or, alternatively, the existence of circular-secure fully homomorphic encryption. Then there exists a non-interactive threshold batch argument for trapdoor threshold languages with statistical soundness. Specifically, for any circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$, if Ext is a trapdoor for the threshold language associated with C , then the common reference string size, proof size and verification time for the threshold BARG are all bounded by $\text{poly}(\lambda, \log k, |C|, |\text{Ext}|)$.*

Prior constructions of sometimes statistically sound succinct non-interactive arguments [DHJQ26, Nas26] relied on iO , and moreover, had a long common reference string that grows polynomially with k . In contrast, Theorem 1.1 can be instantiated from simpler tools and the parameter sizes grow polylogarithmically with k .

Puncturable threshold signatures. Next, we show that our threshold batch argument can be used to build a *puncturable* threshold signature scheme. First, in a vanilla threshold signature scheme, there are k signers, each with their individual signing key $\text{sk}_1, \dots, \text{sk}_k$. In addition, the public parameters contain a (short) verification key vk for the quorum. In a t -out-of- k threshold scheme, if a set S of t signers all sign a message m to produce a collection of signatures $\{\sigma_i\}_{i \in S}$, it is possible to aggregate these signatures into a single, succinct signature σ_{agg} on m that verifies under vk . The unforgeability requirement says that an adversary who possesses fewer than t signatures on a message m should not be able to create a signature on m that verifies with respect to vk .

Now, we say the threshold signature scheme is *puncturable* if for any message m^* and threshold t , it is possible to sample the verification key vk (as well as the individual signing keys for any $(t - 1)$ -subset of the signers) such that there does not exist any aggregate signature on m^* that verifies with respect to vk . This is a direct generalization of the notion of puncturable signatures (also called all-but-one signatures) from [GVW19] to the threshold setting. As we show in Section 7, puncturable threshold signatures follow readily from our BARG for trapdoor threshold languages (in conjunction with other standard primitives). Moreover, the puncturing property allows us to compose with witness encryption to obtain a threshold *encryption* scheme. We summarize our construction in the following theorem statement:

Theorem 1.2 (Puncturable Threshold Signatures; Informal). *Assume either witness encryption and LWE with a sub-exponential modulus-to-noise ratio, or circular-secure fully homomorphic encryption. Then there is a puncturable threshold signature scheme where the sizes of the individual signatures, the aggregate signatures, and the verification key are $\text{poly}(\lambda, \log k)$, where k is the number of signers.*

The natural approach to building a threshold signature from a threshold BARG is to define the aggregation algorithm to be a threshold BARG proof of knowledge of at least t signatures on the message. For instance, we can take $C(i, \cdot)$ to be the circuit that verifies a signature with respect to signer i . The key challenge, however, is designing C so there is also a *compact* trapdoor Ext associated with C that satisfies Equation (1.1). To achieve the succinctness requirement in Theorem 1.2, the size of Ext must be *polylogarithmic* in the number of signers k . Looking ahead, the $\text{Ext}(j_1, j_2)$ algorithm in the construction will essentially need to compute the number of corrupted signers that lie within the interval $[j_1, j_2]$.

To do this, we design a new combinatorial mechanism based on efficiently-sampleable hypergeometric distributions [GGN10]. The sampler enables us to efficiently sample pseudorandom strings of a fixed Hamming weight (which will correspond to the number of corrupted signers). Crucially, this method allows us to efficiently compute the Hamming weight of any contiguous substring of a pseudorandom string. We exploit this in our design of Ext .

Next, we show that any threshold signature scheme can be generically upgraded to a *weighted* threshold signature scheme via a standard powers-of-two transformation. This transformation preserves punctured security.

In addition, if we work in the random oracle model, our puncturable (weighted) threshold signature scheme also supports *silent* setup [MRV⁺21, DCX⁺23, GJM⁺24]. In this model, users sample their own public/private key pair. Then, there is a public, deterministic aggregation algorithm that takes the public keys from any group of users

and aggregates them together to form the verification key associated with that group. This can be viewed as a decentralized or trustless version of threshold signatures.

Weighted threshold encryption. Composing our puncturable weighted threshold signatures with witness encryption [GGSW13] yields a weighted threshold encryption scheme (with non-interactive decryption). As mentioned previously, in order to leverage witness encryption security, we require a strong *statistical* security property on the underlying threshold signature scheme (i.e., when vk is punctured at a message m^* , there does not exist any signature on m^*). Our scheme supports λ -bit weights. We summarize our construction below:

Theorem 1.3 (Weighted Threshold Encryption; Informal). *Assume the existence of witness encryption and the hardness of LWE with a sub-exponential modulus-to-noise ratio. Then there exists a weighted threshold encryption scheme with static security. We support λ -bit weights and the ciphertext size is $\text{poly}(\lambda, \log k)$, where k is the number of users. In addition, if we work in the random oracle model, then we obtain a silent weighted threshold encryption scheme (from the same assumptions).*

As mentioned above, the only prior constructions of weighted threshold encryption that simultaneously support large weights (i.e., λ -bit weights) and non-interactive decryption rely on iO [DJWW25]. In contrast, unweighted threshold encryption is known from a broad range of assumptions (cf. [Des92, CGS97, BD10]). If we consider the silent setup model, then we have unweighted constructions from bilinear maps [GKPW24, WW26], lattice-based assumptions [AGY26, GY26, CWY26], and obfuscation [RSY21, AFMP20, DJWW25]. Among these, only [DJWW25] supports weights (without virtualization).

Statement on AI use. The technical ideas in this paper were developed by the authors without the use of AI. The authors used ChatGPT and Claude to help prepare the manuscript, primarily in the form of copy-editing, proofreading, and improving the presentation of the paper. The authors carefully reviewed all AI-generated outputs and suggestions, and verified the correctness and originality of all of the content in the paper.

2 Technical Overview

We start by surveying our construction of threshold batch arguments with statistical soundness. To do so, we first recall the canonical monotone-policy BARG from [BBK⁺23], specialized to threshold policies. Let $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ be a circuit and let $t \in [0, k]$ be a threshold. The i^{th} instance is a YES instance if there exists a witness $w \in \{0, 1\}^m$ where $C(i, w) = 1$. The statement being proved is that at least t of the k instances are YES instances.

The summation tree. Assume for simplicity that k is a power of two, and let T be a complete binary tree of height $h = \log k$ whose leaves are the k instances. A prover holding witnesses for a set $S \subseteq [k]$ assigns each leaf i the value $v_i = 1$ if $i \in S$ and $v_i = 0$ otherwise. The prover assigns each internal node the sum of the values of its two children. Thus, the value of each node is the number of leaves of S in its subtree, and the value of the root is $|S|$. The threshold is met when the value of the root is at least t . The proof consists of a succinct commitment to these values, together with a batch argument that every local relation in the tree holds.

Hashing the tree. We instantiate the commitment with a somewhere-extractable hash [HW15, CJJ21b] of the values in the binary tree. In a somewhere-extractable hash function, the key can be sampled to statistically bind on an index i^* . In this case, every digest can only be opened at i^* to a single value, and we can extract this value from the digest using a trapdoor. Furthermore, the key computationally hides the index i^* . The batch argument is an index BARG (i.e., a BARG for index languages) [CJJ21b]. In the construction, the prover hashes each level of the tree separately: for every $\ell \in [0, h]$ it hashes the vector of 2^ℓ values in level ℓ into a digest dig_ℓ , and proves a batch of 2^ℓ statements, one per node of that level, with a proof Π_ℓ . The block $x_{\ell,i}$ hashed at the i^{th} position of level $\ell < h$ records the value v of the node together with the values v_L, v_R it claims for its two children: $x_{\ell,i} = (i, v, v_L, v_R)$. For a leaf node, the block is just $x_{h,i} = (i, v)$. The statements proved at each level are the following.

- **Internal nodes.** For a level $\ell < h$ and a node i within that level, the witness consists of the block $x_{\ell,i}$ and an opening with respect to dig_{ℓ} , together with the blocks of its two children and their openings with respect to $\text{dig}_{\ell+1}$. Recall also that the blocks for each node consist of an index i and a value v , and if the node is an internal node, the values v_L, v_R of its children. The statement associated with internal node i is satisfied if (1) each of the three openings verifies (with respect to either dig_{ℓ} or $\text{dig}_{\ell+1}$); (2) the opened indices match the indices associated with the node and those of its children; (3) the values v_L, v_R claimed for the children match those associated with the openings for the child blocks; and (4) $v = v_L + v_R$.
- **Leaf nodes.** For a leaf j , the witness consists of the block $x_{h,j} = (j, v)$, an opening of $x_{h,j}$ with respect to dig_h , together with a witness w for the j^{th} instance. The statement checks that the opening verifies, that $v \in \{0, 1\}$, and that either $v = 0$ or $C(j, w) = 1$. In particular, a leaf j may take the value 1 only if the prover provides a valid witness for the associated statement j .
- **Root node.** The root is the only node of level 0, and the prover announces its value v_{root} in the clear. The level-0 statement carries the additional check that the value associated with the root node is exactly v_{root} .

The proof is v_{root} together with the $h + 1$ digests and the $h + 1$ batch arguments, and the verifier checks that $v_{\text{root}} \geq t$ and that all $h + 1$ proofs verify. Hashing the levels separately incurs an overhead of $h + 1 = \log k + 1$, which is affordable precisely because the tree has logarithmic depth. Completeness is immediate: the values an honest prover computes satisfy every local relation, and the root has value $|S| \geq t$.

Computational soundness. Recall that the proof includes a somewhere-extractable hash of all of the values in the tree, and we can extract the associated value by programming the hash key accordingly. We define this extracted value to be the value the proof associates with a particular node. Now, we say that a node is *over-claimed* if the value the proof associates with it is greater than the actual number of YES instances in its subtree. Suppose the verifier accepts, but there are fewer than t YES instances. Then the root must be over-claimed, since the proof must associate $v_{\text{root}} \geq t$ with the root (otherwise, the proof does not verify and the verifier rejects). The strategy is to propagate this violation down to a leaf, where we will extract a witness for a false instance (which gives a contradiction).

- Suppose the hash key in level ℓ is programmed to be extracting on node u , and let (u, v, v_L, v_R) be the block associated with it. Suppose moreover that the BARG is somewhere sound [CJJ21b]. Namely, we can program a hidden index i^* into the BARG and it is computationally hard for a prover to find an accepting BARG proof where instance i^* is false. Suppose the BARG is programmed to be sound for the instance associated with node u , and moreover, that the proof Π_{ℓ} for level ℓ is valid. Then $v = v_L + v_R$, and there exist openings for $\text{dig}_{\ell+1}$ to the two children of u to blocks with values v_L and v_R .
- The number of YES instances in the subtree of u is the sum of the number of YES instances in the subtrees associated with its two children. If u is over-claimed, then $v = v_L + v_R$ must exceed this sum. Thus, either its left child (with value v_L) or its right child (with value v_R) must also be over-claimed. We can now program the hash key in level $\ell + 1$ to be binding on the associated child. By design, the associated child is now an over-claimed node, and we can appeal to the same argument as before to propagate level-by-level down the tree until we reach a leaf node.
- For a leaf j , both its associated value and the number of YES instances in its subtree are either 0 or 1. Thus, if j is over-claimed, then its associated value must be $v = 1$ and the j^{th} instance is a NO instance. In this case, if the leaf statement (j, v) is satisfied and $v = 1$, the statement (j, v) is true only if there exists w where $C(j, w) = 1$. However, this is a contradiction of the premise that instance j is a NO instance. Thus, if the BARG is computationally sound for instance j , then an efficient prover should not be able to come up with a satisfying proof for the leaf layer.

A subtlety in this analysis is the fact that the set of nodes which are over-claimed is a function of the adversary's proof, but the hash keys that determine which nodes we bind on must be fixed *before* the adversary produces its proof. To get around this, the canonical reduction strategy leverages the fact that the tree is perfectly balanced so $h = \log k$. In this case, it can simply guess a path from the root to a leaf and set the hash keys to be binding along the path. Since there is at least one path that is over-claimed, the guess will be correct with probability $1/2^h = 1/k$,

which is inverse polynomial. This suffices to prove computational soundness. Note here that we *cannot* guess the full set of false statements, since this would incur an exponential loss in k . Complexity leveraging in this setting would blow up all parameters by a factor of k , which violates succinctness.

Statistical soundness. The previous proof strategy does not achieve statistical soundness for two main reasons:

- The underlying index BARG is only computationally sound, which means that we cannot ensure security against an unbounded adversary (i.e., when the BARG is programmed to be sound on an index j , a computationally unbounded prover could still produce an accepting proof even when the j^{th} statement is false). One way to solve this problem is to use a *somewhere-statistically* sound BARG. In this setting, if the BARG is programmed to be statistically sound on index j , even an unbounded prover will not be able to find an accepting proof on a batch of statements where the j^{th} statement is false. Somewhere-statistically sound BARGs can be built from pairings [WW22, CEW25, BFN26], circular-secure fully homomorphic encryption (FHE) [HW26], witness encryption and LWE [NWW26], or *iO* [HW26, Nas26].
- Our soundness analysis sketched above additionally relies on computational steps within it. For instance, in order to program indices into the hash keys or the BARG common reference string, we have to rely on *computational* mode indistinguishability of the underlying primitives.

To address these challenges, we relax our soundness requirements to only require statistical soundness for *trapdoor* languages. Specifically, suppose we have k instances and let T be the binary tree associated with those instances (where the value of each node is the number of YES instances in the subtree rooted at the node). Suppose now that in addition to the circuit C , we are also given a function Ext that assigns an integer to each node of T that satisfies the following two requirements:

- **Leaf soundness:** $\text{Ext}(i) \in \{0, 1\}$ for every leaf i , and if $\text{Ext}(i) = 0$, then the i^{th} instance is a NO instance.
- **Propagation:** $\text{Ext}(u) = \text{Ext}(u_L) + \text{Ext}(u_R)$ for every internal node u with children u_L and u_R .

We refer to such a function as a *trapdoor* for the threshold language associated with C . Unrolling the propagation rule, $\text{Ext}(u)$ counts the number of leaves where Ext outputs 1 in the subtree of u . By leaf soundness, Ext must output 1 on every YES instance. Thus, $\text{Ext}(u)$ always gives an upper bound on the number of YES instances in the subtree rooted at u . In particular, $\text{Ext}(\text{root}) < t$ certifies that the threshold is not met. Now, when building a threshold BARG for trapdoor languages with statistical soundness, we provide a description of Ext as input to the binding setup. When the CRS is sampled in this way, the resulting threshold BARG is statistically sound for all circuits C where Ext is a trapdoor. Finally, we allow the size of the common reference string and the verification time to grow with the description size of Ext , so in applications, the trapdoor function Ext must have a short description. In particular, this means simply hard-wiring the indices of the NO instances into Ext is out of the question.

Consider again our basic proof strategy described above. If Ext is a trapdoor for a circuit C , then we say a node u is over-claimed when the value associated with u exceeds $\text{Ext}(u)$. Thus, given a node u , deciding which of its two children to consider next just requires comparing the extracted values for its children against the values $\text{Ext}(u_L)$ and $\text{Ext}(u_R)$. Instead of having the reduction guess the path from the root to a leaf (associated with a NO instance), the proof can use Ext to determine the path to take. The remaining problem, however, is that we still need to program the hash keys at each layer to bind to a specific index (in order to extract the value associated with the particular block). In vanilla somewhere-extractable hash functions, these indices are fixed at setup time. To carry out our proof strategy, we need a way to sample the hash key in a way that it “dynamically” binds to an index determined *after* the adversary has picked its proof (recall that the adversary’s proof determines the path of over-claimed nodes in T).

Binding dynamically. To carry out the proof strategy, we thus consider *dynamic* versions of somewhere-extractable hash functions and index BARGs. In both cases, we augment the syntax to allow the special index (where the hash function is extracting or the BARG is statistically sound) to be determined *after* the public parameters for the scheme (e.g., the hash key or the CRS) are sampled. Specifically, in a *dynamic* somewhere-extractable hash, the special index where we can extract is *not* determined at setup time, but instead, determined as a function of a tag that is specified *after* the public parameters for the hash function are generated. In more detail, a dynamic somewhere-extractable hash function operates as follows:

- Like standard somewhere-extractable hash functions, we support a normal setup and a binding setup. The binding setup takes as input a *function* $f: \mathcal{T} \rightarrow [k]$, where $\mathcal{T}$ is the tag space and k is the input dimension. As usual, we require the parameters output by the normal setup and the binding setup to be computationally indistinguishable. The binding setup additionally outputs a trapdoor td .
- Next, we introduce an *additional* key-generation algorithm that takes as input the public parameters and a tag $\tau \in \mathcal{T}$, and outputs a hash key hk .
- The hash key can be used like a normal somewhere-extractable hash key in order to hash a vector into a succinct digest and produce succinct local openings for each block.
- The main property for a dynamic somewhere-extractable hash is that in binding mode, a hash key hk associated with tag τ is extractable on the index $f(\tau)$. The main difference with a vanilla somewhere-extractable hash function is the fact that the special index where we can extract is now a (secret) function of a (public) tag τ , as opposed to being fixed as part of the public parameters.

We show how to build a dynamic somewhere-extractable hash function from leveled homomorphic encryption in [Appendix A](#). Our construction closely follows the classic construction of somewhere-statistically binding hash functions from [\[HW15\]](#).

The second primitive we require is a functional index BARG [\[PP26\]](#), which can be thought of as the analogous notion for a BARG. Here, the common reference string for a BARG is programmed with such a function f as well, and proofs are generated with respect to a tag τ . We can extract a witness for the instance $f(\tau)$ from *any* accepting proof. Paneth and Pass [\[PP26\]](#) recently constructed a functional BARG with a weaker *computational* extraction property, which unfortunately does not suffice for our applications. In [Appendix B](#), we show how to construct a functional BARG with *statistical* extraction from a dynamic somewhere-extractable hash function and a somewhere-statistically correlation-intractable hash function for sparse product relations [\[HLR21, HW26\]](#). All of these primitives in turn follow from witness encryption and LWE with a sub-exponential modulus-to-noise ratio or from circular-secure FHE.

Putting it together. Returning now to our threshold BARG construction, our construction follows the canonical construction, except we instantiate the underlying somewhere-extractable hash function and the index BARG with their dynamic variants. The public parameters include the public parameters for the dynamic somewhere-extractable hash function and the functional index BARG.

- As before, given the instances, the prover starts by constructing the tree T where the leaves are associated with the instances, and the value of each node is associated with the number of claimed YES instances in the subtree rooted at that node.
- Next, the prover hashes the values within each layer top-down, where the tag for level $\ell > 0$ is the digest $\text{dig}_{\ell-1}$ associated with the previous level. The root level has no previous level, so its tag is simply the empty tag $\perp$. Note that we have a different (and independently sampled) hash key for each layer of the tree. In the security analysis below, for each level $\ell < h$, we will hard-wire the extraction trapdoor for the hash key at layer ℓ into the function associated with the hash key in layer $\ell + 1$.
- As before, the prover then provides a (functional) index BARG proof for each level of the tree. The tag for the BARG proof at level ℓ is the same as the tag for the hash key at level ℓ . Namely, at the root level, we default to an empty tag $\perp$ and for subsequent layers, the tag is the digest $\text{dig}_{\ell-1}$ for the previous layer.
- The rest of the construction proceeds as before. Namely, the overall proof consists of the value associated with the root node together with the digests and batch arguments associated with each layer of the tree.

To prove statistical soundness in binding mode (with respect to a trapdoor Ext), we sample the public parameters of the dynamic somewhere-extractable hash function and the functional index BARG to bind to a function that uses the trapdoor Ext to dynamically determine the index of the over-claimed node in the given layer. In more detail, we sample the public parameters top-down as follows:

- **Root level.** The root level consists of a single node, so there is a single index to bind on. Thus, we sample the public parameters for the hash function and the functional index BARG for the root level to be binding with respect to the constant function that always outputs 1 (i.e., the index of the root node).
- **Remaining levels.** For a level $\ell > 0$, we sample the public parameters of the hash function and that of the functional index BARG to be binding with respect to the function g_ℓ that takes as input a tag $\text{dig}_{\ell-1}$ from the previous layer. The function g_ℓ has the trapdoor function Ext and the extraction trapdoor $\text{td}_{\ell-1}$ for the hash key at level $\ell - 1$ hard-wired inside it. Then, on input the digest $\text{dig}_{\ell-1}$, the function g_ℓ does the following:
 - Use the trapdoor $\text{td}_{\ell-1}$ to extract the block (u, v, v_L, v_R) from $\text{dig}_{\ell-1}$. This is the block associated with the node u in level $\ell - 1$ on which the hash key at level $\ell - 1$ is binding.
 - Let u_L, u_R denote the indices associated with the left and right children of u in the current layer (recall that u is the index of a node in the previous layer). If $v_L > \text{Ext}(u_L)$, then output the index of u_L . Otherwise, output the index of u_R .

Our main invariant is that if the node u in level $\ell - 1$ is over-claimed, then the hash function and the functional index BARG in level ℓ bind to a child of u that is also over-claimed. Specifically, if u is over-claimed, then $v_L + v_R = v > \text{Ext}(u) = \text{Ext}(u_L) + \text{Ext}(u_R)$. Finally, if the batch of statements is false and the verifier accepts, then $v_{\text{root}} \geq t$. For a batch of false statements, $\text{Ext}(\text{root}) < t$, so the invariant holds at the root layer. This ensures that at the leaf layer, both the dynamic somewhere-extractable hash function and the functional index BARG are binding on an over-claimed leaf node associated with a false statement. However, no such proofs can exist in this case, which proves the claim. We provide the formal construction and analysis in [Section 5](#).

2.1 Puncturable Threshold Signatures and Weighted Threshold Encryption

Our first application of a statistically sound threshold BARG is to threshold signatures. In this setting, there is an (aggregated) verification key vk_{agg} for a group of users called the signing quorum. The signing quorum consists of k users, and each user $i \in [k]$ holds a signing key sk_i . Using their signing key sk_i , a user can sign a message m and produce a signature share σ_i . Finally, anyone can take a collection of signature shares $\{\sigma_i\}_{i \in T}$ together with a (public) aggregation key ak to obtain a signature σ on m that verifies under the quorum’s verification key vk_{agg} . In the threshold setting, aggregation requires a minimum of t signature shares, where t is the threshold. Security for a threshold signature scheme says that an adversary who has fewer than t signature shares on any message m should not be able to forge a signature on m that verifies with respect to vk_{agg} . The main efficiency property is that the size of the aggregate signature should be independent of both the number of signers k and the threshold t .

In the classic setting of threshold signatures [\[Des87\]](#), a central dealer is responsible for generating the verification key vk_{agg} for the quorum as well as the signing keys $\text{sk}_1, \dots, \text{sk}_k$ for each user. More recently, a number of works [\[MRV⁺21, DCX⁺23, GJM⁺24\]](#) have considered threshold signatures in a decentralized setting where individual users each choose their own verification key vk_i and signing key sk_i . Then, there is a public aggregation algorithm that takes as input the public verification keys $\text{vk}_1, \dots, \text{vk}_k$ from a group of users and aggregates them together to obtain a verification key vk_{agg} associated with the particular group. In this case, vk_{agg} functions as a verification key for the quorum and each user’s individual sk_i is their signing key. Such schemes are called threshold signatures with silent setup. Note that while the users all generate their individual verification keys and signing keys independently, we do assume that they sample them with respect to a common reference string.

In this work, we consider a stronger notion of *puncturable* threshold signatures in the decentralized setting. In this setting, the common reference string for the threshold signature scheme can be sampled so that there does not exist *any* signature on a target message m^* . Moreover, the CRS computationally hides the punctured message m^* . Clearly, such a scheme satisfies (selective) unforgeability.¹ We now define our notion more precisely:

- Let k be the size of the signing quorum. We require that there exist a simulator which takes as input a message m^* and a set of “corrupted users” $S \subseteq [k]$. The simulator outputs a common reference string crs for the

¹In the selective unforgeability game, the adversary has to declare the message m^* it will forge on at the beginning of the security game. In this case, if the CRS is punctured on m^* , then there does not exist any signature on m^* , so the adversary cannot win. Thus, punctured security (together with the computational indistinguishability of the punctured CRS) immediately implies selective unforgeability.

threshold signature scheme together with a collection of verification key/signing key pairs (vk_i, sk_i) for the honest users $i \in [k] \setminus S$. Then, with overwhelming probability, we require that for *all* choices of corrupted verification keys $\{vk_i\}_{i \in S}$ and all thresholds $t > |S|$, there does not exist a signature on m^* for the threshold t with respect to the aggregated verification key vk_{agg} associated with the signing quorum $\{vk_i\}_{i \in [k]}$. We note that this is a *statistical* guarantee.

- Moreover, we require that the simulated common reference string and honest verification keys $\{vk_i\}_{i \in [k] \setminus S}$ be computationally indistinguishable from those output by the normal setup and key-generation algorithms. This should hold even for an adversary that obtains signatures from the honest users on messages $m \neq m^*$ of its choosing (signed using the simulated signing keys sk_i).

We build a threshold signature scheme with silent setup from threshold BARGs, digital signatures, and a hash function with succinct local openings (e.g., a Merkle tree [Mer89]). Assume for simplicity that k is a power of two.

- **Setup.** The common reference string crs consists of a hash key hk , a common reference string crs_{BARG} for a threshold BARG, and a random permutation $(j_1, \dots, j_k)$ over $[k]$.² The permutation associates each user $\ell \in [k]$ with an index $j_\ell \in [k]$.
- **Key generation and signing.** The key-generation algorithm simply samples a key pair (vk_ℓ, sk_ℓ) for the digital signature scheme. To sign a message m , user ℓ signs m under sk_ℓ to obtain a signature share σ_ℓ .
- **Key aggregation.** Given a collection of verification keys $(vk_1, \dots, vk_k)$ from the k users, we define $x_{j_\ell} = (j_\ell, vk_\ell)$ for each $\ell \in [k]$. The aggregation algorithm then hashes the vector $\mathbf{x} = (x_1, \dots, x_k)$ under hk to obtain a digest dig along with local openings $\rho_1, \dots, \rho_k$. The aggregated verification key is $vk_{\text{agg}} = (hk, \text{crs}_{\text{BARG}}, \text{dig})$. The aggregation key ak consists of vk_{agg} together with the vector $\mathbf{x}$ and the openings $\rho_1, \dots, \rho_k$.
- **Signature aggregation.** Let C be the circuit with hk , dig , and m hard-wired inside it. The circuit takes as input an index $j \in [k]$ and a witness (vk, σ, ρ) , and outputs 1 if the following two conditions hold:
 - ρ is a valid opening of the digest dig to the pair (j, vk) at index j ; and
 - σ is a valid signature on m under vk .

Given signature shares $\{\sigma_\ell\}_{\ell \in T}$ on m from a set of users $T \subseteq [k]$, the aggregator uses the aggregation key ak to form the witness $w_{j_\ell} = (vk_\ell, \sigma_\ell, \rho_{j_\ell})$ for the j_ℓ^{th} instance, for each $\ell \in T$. It then computes a threshold BARG proof for C using these $|T|$ witnesses. This is the aggregate signature σ_{agg} .

- **Verification.** To verify an aggregate signature σ_{agg} on a message m with respect to a threshold t , the verifier reconstructs the circuit C from vk_{agg} and m , and accepts if σ_{agg} is a valid threshold BARG proof (under crs_{BARG}) attesting that at least t instances of C are YES instances.

Correctness follows from the correctness of the underlying primitives and succinctness follows from the succinctness of the hash function and the threshold BARG.

Security. To argue that the above scheme satisfies punctured security, we need to instantiate the digital signature scheme with an all-but-one signature scheme [GVW19] and the hash function with a function-binding hash function [FWW23]. An all-but-one signature scheme (also called a puncturable signature scheme) is the analog of a puncturable threshold signature scheme adapted to the setting of plain digital signatures. More formally, an all-but-one signature scheme augments a plain digital signature scheme with an additional punctured setup algorithm:

- The punctured setup takes as input a message m^* and outputs a key pair (vk, sk) such that there does not exist *any* accepting signature on m^* under vk .

²For ease of exposition, we include the permutation in the CRS here, which means the CRS grows with (and depends on) the number of users k . In our actual construction (Construction 7.3), the common reference string is short, and the key-aggregation algorithm instead derives the permutation using public randomness. We obtain a silent threshold encryption scheme with a short CRS by working in the random oracle model and using the random oracle to derive the key-aggregation randomness (see Section 10).

- The verification key output by the punctured setup algorithm is computationally indistinguishable from a normal verification key. This is true even if the adversary can adaptively request signatures on messages $m \neq m^*$ under sk .

A function-binding hash function [FWW23] is similar to a somewhere-statistically binding hash function [HW15], except it statistically binds to a function of the input rather than an individual block. Formally, in a function-binding hash function for conjunctions over a block space $\mathcal{X}$, we can sample the hash key hk to bind to a block function $f: \mathcal{X} \rightarrow \{0, 1\}$. Then, suppose dig is the digest of a vector $(x_1, \dots, x_k) \in \mathcal{X}^k$ where $f(x_i) = 1$ for all $i \in [k]$. If hk is binding for the function f , then dig cannot be opened at any index to a block x where $f(x) = 0$. Moreover, a binding hash key is computationally indistinguishable from a normal hash key.

Suppose we instantiate the above threshold signature scheme with a function-binding hash function and an all-but-one signature scheme. We now describe our simulator. Let m^* be the punctured message and $S \subseteq [k]$ be the set of corrupted users. The simulator works as follows:

- For the sake of exposition, suppose that the simulator is able to sample a random permutation $(j_1, \dots, j_k)$ on $[k]$ together with an algorithm $Query$, where $Query(j) = 1$ if index j is associated with a corrupted user and $Query(j) = 0$ otherwise. Moreover, suppose we can efficiently extend the $Query$ algorithm to the internal nodes of the summation tree. Namely, consider the binary tree with k leaves where the j^{th} leaf has value $Query(j)$. Then, for an internal node u of this tree, we require $Query(u)$ to return the number of corrupted users among the leaves in the subtree rooted at u . Later on, we will describe how to efficiently implement the $Query$ algorithm.
- For each honest user $\ell \in [k] \setminus S$, the simulator samples a key pair (vk_ℓ, sk_ℓ) that is punctured on the message m^* . Moreover, we assume that the randomness for the punctured setup is derived from $F(K_{\text{PRF}}, j_\ell)$, where F is a pseudorandom function (PRF) and K_{PRF} is the PRF key.
- The simulator samples hk to be binding on the block function f that checks that a block (j, vk) *either* belongs to a corrupted user (i.e., $Query(j) = 1$) *or* the associated verification key is the punctured key (i.e., vk is obtained by running the punctured setup with randomness $F(K_{\text{PRF}}, j)$). Note that here, the description of the $Query$ algorithm as well as the PRF key K_{PRF} are hard-wired within the block function f .
- Finally, the simulator samples crs_{BARG} for the threshold BARG to be binding with respect to the trapdoor $\text{Ext} = Query$.

Consider an execution of the punctured security game. Take any collection of verification keys $\{vk_i\}_{i \in S}$ for the corrupted users. The aggregated verification key is derived from $(vk_1, \dots, vk_k)$ where vk_i for $i \in [k] \setminus S$ are sampled by the simulator. Consider each index $j \in [k]$:

- If $j = j_\ell$ for a corrupted user $\ell \in S$, then $Query(j) = 1$.
- If $j = j_\ell$ for an honest user $\ell \in [k] \setminus S$, then the simulator samples vk_ℓ to be a punctured key where the randomness is derived from $F(K_{\text{PRF}}, j)$.

In both cases, the block (j, vk_ℓ) satisfies f , so the function-binding property of the hash function ensures that at every index j where $Query(j) = 0$, the digest dig can only be opened to a punctured verification key, which does *not* accept any signature on m^* . Thus, the only YES instances for the circuit C associated with the challenge message m^* are those associated with the indices of the corrupted users. In this case, the $Query$ algorithm serves as a trapdoor for C and $Query(\text{root}) = |S| < t$. Since this holds for every choice of corrupted verification keys, we can appeal to statistical soundness of the threshold BARG to argue that there does not exist an aggregate signature on m^* . This proves punctured security. Finally, we note that mode indistinguishability (i.e., that the simulated CRS and verification keys are computationally indistinguishable from the real CRS and verification keys) follows by a standard hybrid argument using the security of the all-but-one signature scheme, the pseudorandomness of the PRF, and the mode indistinguishability of the hash function and of the threshold BARG.

Implementing Query via pseudorandom hypergeometric sampling. The remaining question is how to implement Query efficiently. Recall that the Query algorithm indicates whether a given index $j \in [k]$ is associated with a corrupted user or not, and moreover, for internal nodes u in the summation tree, $\text{Query}(u)$ returns the number of corrupted users in the subtree rooted at u . A naïve approach is to hard-wire the indices of the corrupted users into the description of Query. However, this means the description length of Query now grows with the number of corrupted users. This is $\Omega(k)$ in the worst case. Since the description of Query is hard-wired into both the BARG trapdoor and the block function f for the function-binding hash function, this means the size of crs_{BARG} and hk (and thus, of vk_{agg}), as well as the verification time, would all grow with the total number of users k . The resulting threshold signature scheme is no longer succinct in this case.

Our goal is to design a scheme where we can implement the Query algorithm with a circuit whose size scales only polylogarithmically with the number of users. This means we need a way to build a compact representation of the indices of the corrupted users. To do so, observe first that the output of Query only requires knowledge of the indices of the corrupted users. This means it only depends on the indicator string $x \in \{0, 1\}^k$ associated with these positions (i.e., the string where $x_j = 1$ if the j^{th} index is corrupted and $x_j = 0$ otherwise). This is a string of Hamming weight $c = |S|$. Moreover, the *position* of the non-zero elements in x is uniformly random over $[k]$ (over the choice of the random permutation $(j_1, \dots, j_k)$ that the simulator samples). Thus, an equivalent way of sampling the random permutation is to first sample a random vector $x \in \{0, 1\}^k$ with Hamming weight exactly c , and then randomly assign the corrupted users to the indices $j \in [k]$ where $x_j = 1$ and randomly assign the honest users to the indices $j \in [k]$ where $x_j = 0$.

Next, instead of sampling the string x directly, we describe an approach to sample it in a top-down manner. Consider the summation tree. The value of the root node corresponds to the number of corrupted indices among all k indices. This is fixed to be $c = |S|$. Some number c_L of the corrupted indices will fall in the left subtree while the remainder c_R will fall in the right subtree. The decision then is how to split c into c_L and c_R such that $c_L + c_R = c$. When the indices are assigned randomly (as they are in the real scheme), the distribution of c_L is precisely a *hypergeometric distribution* ($\text{HGD}_{1/2}$). We can sample c_L from the hypergeometric distribution in time $\text{poly}(\lambda, \log k)$ with $2^{-\lambda}$ statistical error [GGN10] and then set $c_R = c - c_L$. We can now repeat this process for each internal node layer by layer until we reach the leaves. We use this process to define the permutation $(j_1, \dots, j_k)$. The distribution of $(j_1, \dots, j_k)$ associated with this hypergeometric sampling procedure is statistically close to that obtained using a random permutation. Finally, we de-randomize the sampling procedure by deriving the randomness for the hypergeometric sampling from a PRF. Observe that this immediately yields an efficient algorithm to implement the Query function: start from the root, and recursively compute the hypergeometric samples all the way down to the parent of the target node. This requires at most $\log k$ PRF evaluations and invocations of the hypergeometric sampler. For ease of exposition, in Section 6, we introduce the notion of a *pseudorandom fixed-weight string* to capture this functionality.

Supporting weighted thresholds. In a weighted threshold signature scheme, each signer is additionally associated with a weight w_ℓ , and an aggregate signature attests that the signers have a total weight of at least t . One approach to support weights using an unweighted threshold signature scheme is through virtualization where we implement a signer with weight w_ℓ with w_ℓ virtual signers in the underlying unweighted scheme. However, this incurs a cost that scales with the *magnitude* of the weights, so such a scheme can only support polynomial-size weights. A natural goal is to support arbitrary weights where the cost scales with the *bit-length* of the weights rather than the magnitude.

Here, we observe that there is a simple powers-of-two transformation to transform any unweighted threshold signature scheme into a weighted one. Suppose we want to support L -bit weights. We write the weight of each user ℓ in binary as $w_\ell = \sum_{j \in [0, L-1]} 2^j w_{\ell,j}$, and run L independent copies of the unweighted scheme, one for each bit position. We will treat a signature in the j^{th} scheme as having weight 2^j .

- Each user $\ell \in [k]$ samples a verification/signing key pair $(\text{vk}_{\ell,j}, \text{sk}_{\ell,j})$ for each of the L schemes. The user publishes $\text{vk}_\ell = (\text{vk}_{\ell,0}, \dots, \text{vk}_{\ell,L-1})$. When signing, the user signs the message under each of its signing keys (so each signature share consists of L underlying signatures).
- To aggregate a collection of verification keys $(\text{vk}_1, \dots, \text{vk}_k)$ with the associated weights $(w_1, \dots, w_k)$, the aggregator first defines $I_j \subseteq [k]$ to be the set of users ℓ where $w_{\ell,j}$ is 1. Let $k_j = |I_j|$. The aggregated verification key consists of L aggregated verification keys for the underlying schemes, where the j^{th} aggregated verification key is derived from $\{\text{vk}_{\ell,j}\}_{\ell \in I_j}$. We also include the counts $k_0, \dots, k_{L-1}$ as part of the aggregated verification key.

- Given signature shares from a set of users $T \subseteq [k]$, the aggregate signature consists of L signatures. The j^{th} aggregate signature consists of the count $c_j = |T \cap I_j|$ together with an aggregate signature for copy j with respect to the threshold c_j .
- The verifier checks that the aggregate signature for each copy j is valid with respect to c_j , that $c_j \leq k_j$, and that $\sum_{j \in [0, L-1]} 2^j c_j \geq t$.

Correctness follows from the fact that the total weight of the users in T is $\sum_{j \in [0, L-1]} 2^j |T \cap I_j|$. Succinctness follows from the fact that the parameter sizes of all components only grow by a factor of L , which is *polylogarithmic* in the total weight. Security and mode indistinguishability follow from the corresponding properties of the underlying scheme and we refer to [Section 8](#) for the full details.

Weighted threshold encryption. Finally, we observe that we can directly lift a puncturable weighted threshold signature scheme into a weighted threshold encryption scheme by composing with witness encryption [GGSW13]. Recall first that in weighted threshold encryption (with silent setup), each user generates their own public/private key pair (independently of all other users). Then, there is a public aggregation algorithm that takes a collection of user public keys along with the associated weights and aggregates them into a short public encryption key ek . Anyone can use ek to encrypt a message μ with respect to a weight threshold t , and any set of users with total weight at least t can decrypt by each publishing a decryption hint. To obtain a threshold encryption scheme from a puncturable threshold signature scheme, we proceed as follows:

- The users' keys are signature keys, and the encryption key ek is the aggregated verification key vk_{agg} .
- To encrypt μ , the encrypter samples a fresh key pair $(vk_{\text{OTS}}, sk_{\text{OTS}})$ for a one-time signature scheme, encrypts μ using witness encryption with respect to the NP statement “there exists an aggregate signature on vk_{OTS} under vk_{agg} for the threshold t ,” and signs the resulting ciphertext using sk_{OTS} .
- To generate a decryption hint for a ciphertext, the user first checks the one-time signature, and if it verifies, then the user gives out a signature share on vk_{OTS} .
- To decrypt, one aggregates the signature shares on vk_{OTS} and uses the aggregate signature as the witness to decrypt the witness encryption ciphertext.

To argue security, we puncture the threshold signature scheme on the one-time signature key vk_{OTS}^* associated with the challenge ciphertext. Since the corrupted users have total weight less than t , the challenge statement is now false, which allows us to appeal to the security of the witness encryption scheme and conclude that the message μ is computationally hidden. Here, it is critical that *no* signatures exist on vk_{OTS}^* rather than just being computationally hard to find. This is because invoking witness encryption security requires the underlying NP statement to be false (rather than it just being computationally hard to find a witness). Here, the one-time signature ensures that the adversary cannot request honest decryption hints on vk_{OTS}^* (i.e., signature shares on the punctured message). This is because honest decrypters would only sign vk_{OTS} after checking the one-time signature. Since the adversary in the security game is not allowed to issue decryption queries on the challenge ciphertext, any adversary that makes a valid decryption query on a ciphertext associated with vk_{OTS}^* must necessarily have forged a signature under vk_{OTS}^* . This use of one-time signatures is a standard approach used for building non-malleable or CCA-secure public-key encryption (cf., [DDN91, Sah99]). Succinctness follows from succinctness of the underlying threshold signature scheme (and specifically, succinctness of the aggregated verification key, the aggregate signature, and the size of the verification circuit). We provide the full details in [Section 9](#).

3 Preliminaries

We write λ for the security parameter. All logarithms are base 2. We write $\mathbb{N} := \{0, 1, 2, \dots\}$ for the set of nonnegative integers. For an integer $n \in \mathbb{N}$, we write $[n] := \{1, \dots, n\}$, with the convention that $[0] := \emptyset$, and for integers a, b we write $[a, b] := \{a, a+1, \dots, b\}$, with the convention that $[a, b] := \emptyset$ when $a > b$. Indices are encoded in binary offset by one: for $n \geq 1$, we identify an index $j \in [n]$ with the $\lceil \log n \rceil$ -bit binary representation of $j-1 \in [0, n-1]$,

so an index in $[n]$ occupies exactly $\lceil \log n \rceil$ bits. For a binary string $x \in \{0, 1\}^*$, we denote the length of x by $|x|$. The *size* of a Boolean circuit is its number of gates, counting its input gates; in particular, a circuit of size at most s has at most s input wires. The *depth* of a Boolean circuit is the number of gates on a longest directed path from an input gate to an output gate; in particular, the depth of a circuit is at most its size. We fix a canonical encoding of Boolean circuits as binary strings, and we assume throughout that the encoding is such that the description size of a circuit is at least its size; in particular, a circuit whose description has size at most s has size at most s . We write $\text{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all constants $c \in \mathbb{N}$ and $\text{poly}(\lambda)$ to denote a function that is $O(\lambda^c)$ for some positive constant c . For any set S , we write $x \xleftarrow{R} S$ to denote a uniform random sample x from S . Given two circuits $C_0, C_1: \{0, 1\}^n \rightarrow \{0, 1\}^m \cup \{\perp\}$, we say that C_0 and C_1 are functionally equivalent if $C_0(x) = C_1(x)$ for all $x \in \{0, 1\}^n$. We say that C_b is a *null* circuit if $C_b(x) = \perp$ for all $x \in \{0, 1\}^n$. We denote a trivial null circuit by $C_\perp$. Throughout this work, we say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input.

3.1 Cryptographic Primitives

We now recall the standard definitions of the cryptographic building blocks we use in this work.

Definition 3.1 (Pseudorandom Function [GGM84]). A pseudorandom function (PRF) consists of a pair of efficient algorithms $\Pi_{\text{PRF}} = (\text{Setup}, \text{Eval})$ with the following syntax:

- $\text{Setup}(1^\lambda, N, 1^{\ell_{\text{out}}}) \rightarrow K$: On input a security parameter λ , an input domain size N (in binary), and an output length ℓ_{out} , the key-generation algorithm outputs a key K . We assume the key contains an implicit description of ℓ_{in} and ℓ_{out} .
- $\text{Eval}(K, x) \rightarrow y$: On input a key K and an input $x \in [N]$, the evaluation algorithm outputs a value $y \in \{0, 1\}^{\ell_{\text{out}}}$.

In addition, we require that Π_{PRF} satisfies pseudorandomness. Formally, for a bit $b \in \{0, 1\}$ and a security parameter λ , we define the pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:

- On input the security parameter 1^λ , the adversary outputs an input domain size N and an output length $1^{\ell_{\text{out}}}$.
- The challenger samples a key $K \leftarrow \text{Setup}(1^\lambda, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}})$.
- The adversary can make an arbitrary number of *distinct* evaluation queries:
 - On each evaluation query, the adversary chooses an input $x \in [N]$.
 - The challenger samples $y_0 \xleftarrow{R} \{0, 1\}^{\ell_{\text{out}}}$ and computes $y_1 = \text{Eval}(K, x)$. It replies to $\mathcal{A}$ with y_b .
- At the end of the game, the adversary outputs a guess $b' \in \{0, 1\}$, which is the output of the experiment.

We say that Π_{PRF} satisfies pseudorandomness if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

Definition 3.2 (One-Time Digital Signatures). A one-time digital signature scheme for arbitrary-length messages is a tuple of efficient algorithms $\Pi_{\text{OTS}} = (\text{Gen}, \text{Sign}, \text{Verify})$ with the following syntax:

- $\text{Gen}(1^\lambda) \rightarrow (\text{vk}, \text{sk})$: On input the security parameter λ , the generation algorithm outputs a verification key vk and a signing key sk .
- $\text{Sign}(\text{sk}, m) \rightarrow \sigma$: On input a signing key sk and a message $m \in \{0, 1\}^*$, the signing algorithm outputs a signature σ .
- $\text{Verify}(\text{vk}, m, \sigma) \rightarrow b$: On input a verification key vk , a message $m \in \{0, 1\}^*$, and a signature σ , the verification algorithm outputs a bit $b \in \{0, 1\}$.

The signature scheme should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all $m \in \{0, 1\}^*$, it holds that

$$\Pr \left[\text{Verify}(\text{vk}, m, \sigma) = 1 \quad : \quad \begin{array}{l} (\text{vk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda) \\ \sigma \leftarrow \text{Sign}(\text{sk}, m) \end{array} \right] = 1.$$

- **Strong one-time unforgeability:** For any adversary $\mathcal{A}$, the strong one-time unforgeability game $\text{ExptOTU}_{\mathcal{A}}(\lambda)$ is defined as follows:

1. On input a security parameter 1^λ , the challenger samples a key pair $(\text{vk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda)$ and sends vk to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ makes at most one signing query: it may send a single message $m \in \{0, 1\}^*$ to the challenger, or it may make no signing query at all.
3. If $\mathcal{A}$ sent a message m , the challenger replies with a signature $\sigma \leftarrow \text{Sign}(\text{sk}, m)$.
4. The adversary outputs m^*, σ^* .
5. The output of the experiment is 1 if $\text{Verify}(\text{vk}, m^*, \sigma^*) = 1$ and, in case $\mathcal{A}$ made a signing query, also $(m^*, \sigma^*) \neq (m, \sigma)$.

We say that Π_{OTS} satisfies strong one-time unforgeability if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that $\Pr[\text{ExptOTU}_{\mathcal{A}}(\lambda) = 1] \leq \text{negl}(\lambda)$.

Definition 3.3 (All-But-One Signatures [GVW19]). An all-but-one (ABO) signature scheme with message space $\{0, 1\}^n$ and signature space $\{0, 1\}^\ell$, where $n = n(\lambda)$ and $\ell = \ell(\lambda)$ are fixed polynomials, is a tuple of efficient algorithms $\Pi_{\text{ABO}} = (\text{Setup}, \text{SetupPunc}, \text{Sign}, \text{Verify})$ with the following syntax:

- $\text{Setup}(1^\lambda) \rightarrow (\text{sk}, \text{vk})$: On input a security parameter λ , the setup algorithm outputs a signing key sk and a verification key vk .
- $\text{SetupPunc}(1^\lambda, \text{msg}^*) \rightarrow (\text{sk}, \text{vk})$: On input a security parameter λ and a message $\text{msg}^* \in \{0, 1\}^n$, the punctured setup algorithm outputs a signing key sk and a verification key vk that is punctured at msg^* .
- $\text{Sign}(\text{sk}, \text{msg}) \rightarrow \sigma$: On input a signing key sk and a message $\text{msg} \in \{0, 1\}^n$, the signing algorithm outputs a signature $\sigma \in \{0, 1\}^\ell$.
- $\text{Verify}(\text{vk}, \text{msg}, \sigma) \rightarrow b$: On input a verification key vk , a message $\text{msg} \in \{0, 1\}^n$, and a signature σ , the verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.

The Π_{ABO} scheme should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, all messages $\text{msg} \in \{0, 1\}^n$, all (sk, vk) in the support of $\text{Setup}(1^\lambda)$, and all σ in the support of $\text{Sign}(\text{sk}, \text{msg})$, the following holds:

$$\text{Verify}(\text{vk}, \text{msg}, \sigma) = 1.$$

- **Punctured correctness:** For all $\lambda \in \mathbb{N}$, all messages $\text{msg}^* \in \{0, 1\}^n$, all (sk, vk) in the support of the punctured setup $\text{SetupPunc}(1^\lambda, \text{msg}^*)$, and all signatures $\sigma \in \{0, 1\}^\ell$, the following holds:

$$\text{Verify}(\text{vk}, \text{msg}^*, \sigma) = 0.$$

- **Verification key indistinguishability:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the verification key indistinguishability game as follows:

1. On input the security parameter 1^λ , the adversary $\mathcal{A}$ outputs a message $\text{msg}^* \in \{0, 1\}^n$.
2. The challenger computes the keys as follows:
 - If $b = 0$, it samples $(\text{sk}, \text{vk}) \leftarrow \text{Setup}(1^\lambda)$.
 - If $b = 1$, it samples $(\text{sk}, \text{vk}) \leftarrow \text{SetupPunc}(1^\lambda, \text{msg}^*)$.

It sends vk to $\mathcal{A}$.

3. Throughout the game, the adversary may adaptively make signing queries: on a query $msg \in \{0, 1\}^n$ with $msg \neq msg^*$, the challenger replies with $\sigma \leftarrow \text{Sign}(sk, msg)$.
4. The adversary outputs a guess b' , which is the output of the experiment.

We say that Π_{ABO} satisfies verification key indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

Definition 3.4 (Witness Encryption [GGSW13]). A witness encryption scheme consists of two efficient algorithms $\Pi_{\text{WE}} = (\text{Enc}, \text{Dec})$ with the following syntax:

- $\text{Enc}(1^\lambda, C, m) \rightarrow \text{ct}$: On input a security parameter λ , a Boolean circuit $C: \{0, 1\}^n \rightarrow \{0, 1\}$, and a message m , the encryption algorithm outputs a ciphertext ct .
- $\text{Dec}(\text{ct}, w) \rightarrow m'$: On input a ciphertext ct and a witness $w \in \{0, 1\}^n$, the decryption algorithm outputs a message m' or a special symbol $\perp$.

The witness encryption scheme should satisfy the following properties:

- **Correctness**: For all $\lambda \in \mathbb{N}$, all Boolean circuits $C: \{0, 1\}^n \rightarrow \{0, 1\}$, all $w \in \{0, 1\}^n$ such that $C(w) = 1$, all messages m , and all ciphertexts ct in the support of $\text{Enc}(1^\lambda, C, m)$, the following holds:

$$\text{Dec}(\text{ct}, w) = m.$$

- **Message hiding**: For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:

1. On input a security parameter 1^λ , the adversary $\mathcal{A}$ sends a Boolean circuit $C: \{0, 1\}^n \rightarrow \{0, 1\}$ and two equal-length messages m_0, m_1 to the challenger.
2. If there exists a witness $w \in \{0, 1\}^n$ such that $C(w) = 1$, the experiment aborts and outputs a random bit $b' \xleftarrow{\mathcal{R}} \{0, 1\}$.
3. The challenger computes a ciphertext $\text{ct} \leftarrow \text{Enc}(1^\lambda, C, m_b)$ and sends ct to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

We say that the witness encryption scheme satisfies message hiding if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

Definition 3.5 (Function-Binding Hash for Conjunctions [FWW23]). Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of block spaces. As in Definition 4.1, the block space is indexed by λ alone and not by the number of blocks k ; an instantiation whose blocks carry an index in $[k]$ takes that component to range over $[2^\lambda]$, and the predicate g below must then be defined on all of $\mathcal{X}_\lambda$. A *function-binding hash* for conjunctions over $\mathcal{X}$ is a tuple of efficient algorithms $\Pi_{\text{FBH}} = (\text{Setup}, \text{SetupBinding}, \text{Hash}, \text{Verify})$ with the following syntax:

- $\text{Setup}(1^\lambda, k, 1^{s_g}) \rightarrow \text{hk}$: On input the security parameter $\lambda \in \mathbb{N}$, a number of blocks $k \in \mathbb{N}$ (in binary), and a description size bound $s_g \in \mathbb{N}$, the setup algorithm outputs a hash key hk .
- $\text{SetupBinding}(1^\lambda, k, 1^{s_g}, g) \rightarrow \text{hk}$: On input the security parameter $\lambda \in \mathbb{N}$, a number of blocks $k \in \mathbb{N}$ (in binary), a description size bound $s_g \in \mathbb{N}$, and (the description of) a predicate $g: \mathcal{X}_\lambda \rightarrow \{0, 1\}$ of description size at most s_g , the binding setup algorithm outputs a hash key hk that binds on the conjunction $\mathbf{x} \mapsto \bigwedge_{i \in [k]} g(x_i)$.

- $\text{Hash}(\text{hk}, \mathbf{x}) \rightarrow (\text{dig}, \rho_1, \dots, \rho_k)$: On input a hash key hk and a vector of blocks $\mathbf{x} = (x_1, \dots, x_k) \in \mathcal{X}_\lambda^k$, the hash algorithm outputs a digest dig and k openings $\rho_1, \dots, \rho_k$. This algorithm is deterministic.
- $\text{Verify}(\text{hk}, \text{dig}, i, x, \rho) \rightarrow b$: On input a hash key hk , a digest dig , an index $i \in [k]$, a block $x \in \mathcal{X}_\lambda$, and an opening ρ , the verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.

Moreover, Π_{FBH} should satisfy the following properties:

- **Correctness:** For all $\lambda, k, s_g \in \mathbb{N}$ with $\lambda, k \geq 1$, all predicates g of description size at most s_g , all hk in the support of $\text{Setup}(1^\lambda, k, 1^{s_g})$ or of $\text{SetupBinding}(1^\lambda, k, 1^{s_g}, g)$, all $\mathbf{x} = (x_1, \dots, x_k) \in \mathcal{X}_\lambda^k$ and all $i \in [k]$, we have $\text{Verify}(\text{hk}, \text{dig}, i, x_i, \rho_i) = 1$, where $(\text{dig}, \rho_1, \dots, \rho_k) = \text{Hash}(\text{hk}, \mathbf{x})$.
- **Perfect function binding:** For all $\lambda, k, s_g \in \mathbb{N}$ with $\lambda, k \geq 1$, all predicates g of description size at most s_g , all hk in the support of $\text{SetupBinding}(1^\lambda, k, 1^{s_g}, g)$, and all $\mathbf{x} = (x_1, \dots, x_k) \in \mathcal{X}_\lambda^k$ such that $g(x_i) = 1$ for every $i \in [k]$, writing $(\text{dig}, \rho_1, \dots, \rho_k) = \text{Hash}(\text{hk}, \mathbf{x})$, the following holds for every index $i \in [k]$, every block $x \in \mathcal{X}_\lambda$ and every opening ρ :

$$\text{Verify}(\text{hk}, \text{dig}, i, x, \rho) = 1 \implies g(x) = 1.$$

- **Mode indistinguishability:** For any efficient adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:
 1. On input the security parameter 1^λ , the adversary $\mathcal{A}$ sends a number of blocks $k \in \mathbb{N}$, a description size bound 1^{s_g} , and the description of a predicate $g: \mathcal{X}_\lambda \rightarrow \{0, 1\}$ of description size at most s_g to the challenger.
 2. If $b = 0$, the challenger samples $\text{hk} \leftarrow \text{Setup}(1^\lambda, k, 1^{s_g})$; if $b = 1$, it samples $\text{hk} \leftarrow \text{SetupBinding}(1^\lambda, k, 1^{s_g}, g)$. It sends hk to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

We say that Π_{FBH} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Succinctness:** There exists a universal polynomial $\text{poly}(\cdot)$ such that for all $\lambda, k, s_g \in \mathbb{N}$ with $\lambda, k \geq 1$, all predicates g of description size at most s_g , all hk in the support of $\text{Setup}(1^\lambda, k, 1^{s_g})$ or of $\text{SetupBinding}(1^\lambda, k, 1^{s_g}, g)$, all $\mathbf{x} \in \mathcal{X}_\lambda^k$ and all $i \in [k]$, we have $|\text{hk}|, |\text{dig}|, |\rho_i| \leq \text{poly}(\lambda, \log k, s_g)$, where $(\text{dig}, \rho_1, \dots, \rho_k) = \text{Hash}(\text{hk}, \mathbf{x})$.

Theorem 3.6 (Function-Binding Hash for Conjunctions [FWW23]). *Assume there exists a leveled homomorphic encryption scheme. Then for every family of block spaces $\mathcal{X}$ there exists a function-binding hash for conjunctions over $\mathcal{X}$.*

4 Building Blocks

In this section, we define two building blocks that we use: dynamic somewhere-extractable hash and statistical functional BARG. The common theme between the two primitives is that they extend known primitives with a “somewhere-statistical property” to have that “somewhere” chosen dynamically.

4.1 Dynamic Somewhere-Extractable Hash

We define a generalization of somewhere-extractable hash [HW15] that enables dynamic selection of the bound index. Namely, the setup algorithm in binding mode receives a function $f: \mathcal{T} \rightarrow [2^\lambda]$, which maps a tag $\tau \in \mathcal{T}$ to an index $i^* = f(\tau)$. This setup algorithm outputs public parameters pp that can be used to *deterministically* generate a hash key hk from any tag. The statistical binding (or extraction) security requires that the hash key $\text{hk} = \text{Gen}(\text{pp}, \tau)$ statistically binds (or is statistically extractable) on the index i^* . We additionally require mode indistinguishability which asserts that a bounded adversary cannot distinguish the parameters sampled by a binding setup from the parameters sampled in normal mode.

Definition 4.1 (Dynamic Somewhere-Extractable Hash). For every $\lambda \in \mathbb{N}$, let $\mathcal{X}_\lambda := \{0, 1\}^n$ be a block space with polynomial-size description, and let $\mathcal{T}_\lambda$ be a tag space. When λ is clear from context, we abuse notation and write $x \in \mathcal{X}$ to refer to $x \in \mathcal{X}_\lambda$ and $\tau \in \mathcal{T}$ to refer to $\tau \in \mathcal{T}_\lambda$. A *dynamic somewhere-extractable hash* for block space $\mathcal{X}$ and tag space $\mathcal{T}$ is a tuple of efficient algorithms $\Pi_{\text{DSEH}} = (\text{Setup}, \text{SetupBinding}, \text{Gen}, \text{Hash}, \text{Verify}, \text{Extract})$ with the following syntax:

- $\text{Setup}(1^\lambda, k, 1^{s_f}, 1^{s_d}) \rightarrow \text{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the number of blocks $k \in \mathbb{N}$ (in binary), a description size bound $s_f \in \mathbb{N}$, and a depth bound $s_d \in \mathbb{N}$, the setup algorithm outputs the public parameters pp.
- $\text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f) \rightarrow (\text{pp}, \text{td})$: On input the security parameter λ , the number of blocks $k \in \mathbb{N}$ (in binary), a description size bound $s_f \in \mathbb{N}$, a depth bound $s_d \in \mathbb{N}$, and (the description of) a function $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d , the binding setup algorithm outputs the public parameters pp and a trapdoor td.
- $\text{Gen}(\text{pp}, \tau) \rightarrow \text{hk}$: On input the public parameters pp and a tag $\tau \in \mathcal{T}$, the key-generation algorithm outputs a hash key hk. This algorithm is deterministic.
- $\text{Hash}(\text{hk}, (x_1, \dots, x_k)) \rightarrow (\text{dig}, \sigma_1, \dots, \sigma_k)$: On input a hash key hk and a vector of blocks $(x_1, \dots, x_k) \in \mathcal{X}^k$, where k is the number of blocks fixed at setup, the hash algorithm outputs a digest dig and openings $\sigma_1, \dots, \sigma_k$.
- $\text{Verify}(\text{hk}, \text{dig}, i, x, \sigma) \rightarrow b'$: On input a hash key hk, a digest dig, an index $i \in \mathbb{N}$, a block $x \in \mathcal{X}$, and an opening σ , the verification algorithm outputs a bit $b' \in \{0, 1\}$.
- $\text{Extract}(\text{td}, \text{dig}) \rightarrow x$: On input the trapdoor td and a digest dig, the deterministic extraction algorithm outputs a block $x \in \mathcal{X}$.

Moreover, Π_{DSEH} should satisfy the following properties:

- **Correctness:** For all $\lambda, s_f, s_d, k \in \mathbb{N}$ with $\lambda, k \geq 1$, all tags $\tau \in \mathcal{T}$, all vectors $(x_1, \dots, x_k) \in \mathcal{X}^k$, and all indices $i \in [k]$, we have:

$$\Pr \left[\text{Verify}(\text{hk}, \text{dig}, i, x_i, \sigma_i) = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda, k, 1^{s_f}, 1^{s_d}) \\ \text{hk} = \text{Gen}(\text{pp}, \tau) \\ (\text{dig}, \sigma_1, \dots, \sigma_k) \leftarrow \text{Hash}(\text{hk}, (x_1, \dots, x_k)) \end{array} \right] = 1.$$

- **Mode indistinguishability:** For any adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:

1. On input a security parameter 1^λ , the adversary $\mathcal{A}$ sends the number of blocks $k \in \mathbb{N}$, a description size bound 1^{s_f} , a depth bound 1^{s_d} and the description of a function $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d to the challenger.
2. The challenger computes the public parameters as follows:
 - If $b = 0$, the challenger computes $\text{pp} \leftarrow \text{Setup}(1^\lambda, k, 1^{s_f}, 1^{s_d})$.
 - If $b = 1$, the challenger computes $(\text{pp}, \text{td}) \leftarrow \text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f)$.

The challenger sends pp to $\mathcal{A}$.

3. The adversary $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

We say that Π_{DSEH} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Somewhere extraction:** For all $\lambda, s_f, s_d, k \in \mathbb{N}$ with $\lambda, k \geq 1$, all functions $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d , all tags $\tau \in \mathcal{T}$, all pairs (pp, td) in the support of $\text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f)$, and the hash key $\text{hk} = \text{Gen}(\text{pp}, \tau)$, the following holds perfectly for *any* (not necessarily honestly computed) digest dig: for any block $x \in \mathcal{X}$ and any opening σ , if $\text{Verify}(\text{hk}, \text{dig}, f(\tau), x, \sigma) = 1$ then $x = \text{Extract}(\text{td}, \text{dig})$.

- **Succinctness:** There exist *efficiently computable* universal polynomials $p_{\text{hk}}(\cdot), p_{\text{dig}}(\cdot), p_{\sigma}(\cdot), p_{\text{Extract}}(\cdot), d_{\text{Extract}}(\cdot)$ that are independent of the tag space $\mathcal{T}$, such that for all $\lambda, s_f, s_d, k \in \mathbb{N}$ with $\lambda, k \geq 1$ and all $\tau \in \mathcal{T}$, the following bounds hold:
 - **Succinct hash key:** For all pp in the supports of $\text{Setup}(1^\lambda, k, 1^{s_f}, 1^{s_d})$ and $\text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, \cdot)$, we have $|\text{hk}| \leq p_{\text{hk}}(\lambda, \log k, s_d, \log s_f)$, where $\text{hk} = \text{Gen}(\text{pp}, \tau)$.
 - **Succinct digest and openings:** For all pp in the support of $\text{Setup}(1^\lambda, k, 1^{s_f}, 1^{s_d})$, all $\mathbf{x} \in \mathcal{X}^k$ and all $i \in [k]$, we have $|\text{dig}| \leq p_{\text{dig}}(\lambda, \log k, s_d, \log s_f)$ and $|\sigma_i| \leq p_{\sigma}(\lambda, \log k, s_d, \log s_f)$, where $\text{hk} = \text{Gen}(\text{pp}, \tau)$ and $(\text{dig}, \sigma_1, \dots, \sigma_k) = \text{Hash}(\text{hk}, \mathbf{x})$.
 - **Succinct trapdoor:** For all functions $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d , and all pairs (pp, td) in the support of $\text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f)$, the function $\text{Extract}(\text{td}, \cdot)$ is computable by a circuit of size $p_{\text{Extract}}(\lambda, \log k, s_d, \log s_f)$ and depth $d_{\text{Extract}}(\lambda, \log k, \log s_d, \log s_f)$.

While succinctness for a somewhere statistically binding hash function [HW15] and similar hash functions usually does not explicitly state a dependence on the depth, we will actually rely on it in our applications.

The construction of this primitive follows by a simple extension of the classic somewhere statistically binding hash function [HW15], and we defer it to [Appendix A](#).

Remark 4.2 (Alternative Syntax for Dynamic Somewhere-Extractable Hash Functions). Alternatively, we could have folded the Gen algorithm into the Hash and Verify algorithms. Namely, the public parameters pp output by Setup and SetupBinding could have been treated as the hash key, and then both Hash and Verify would take the tag τ as additional input, and then derive the actual hash key internally by running $\text{Gen}(\text{pp}, \tau)$. This alternative definition, while more natural, has several downsides:

1. First, it would require deriving the hash key each time a hashing or verification process is invoked, whereas [Definition 4.1](#) enables generating the hash key once, and then reusing it.
2. Second, and perhaps more importantly, one could imagine applications where the function description is much larger than the description of an index in $[2^\lambda]$. The public parameters have to grow with the function description, whereas the hash key derived from $\text{Gen}(\text{pp}, \tau)$ could be independent of the function. Therefore our definition results in potentially smaller hash keys.

Our applications could all be carried out with the alternative definition, despite its downsides. However, we see value in the flexible interface of [Definition 4.1](#).

4.2 Statistical Functional Non-Interactive Batch Arguments

We recall the notion of a functional batch argument (BARG) [BCJP24, PP26] for the index language, which generalizes the notion of a somewhere-extractable index BARG. In an index BARG [CJJ21a, CJJ21b, WW22], an instance is a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$, and a proof establishes that there exist witnesses $w_1, \dots, w_k \in \{0, 1\}^m$ such that $C(i, w_i) = 1$ for all $i \in [k]$, while the proof size and the verification time grow with $\log k$ rather than with k . A somewhere-extractable BARG provides an alternative, computationally indistinguishable, setup that inputs an index $i^* \in [k]$ and outputs a trapdoor along with the common reference string, so that a witness w satisfying $C(i^*, w) = 1$ can be extracted from an accepting proof.

Similar to a dynamic somewhere-extractable hash function, in a functional BARG, the binding setup takes a function $f: \mathcal{T} \rightarrow [k]$ in place of a fixed index, each common reference string is derived from a tag $\tau \in \mathcal{T}$, and extraction is guaranteed on the index $f(\tau)$. As in [Definition 4.1](#), we split setup from the generation of the common reference string: the setup algorithms fix the function f and output public parameters pp , and $\text{Gen}(\text{pp}, \tau)$ deterministically derives the common reference string used by the prover and the verifier. This inherits the first advantage recorded for [Definition 4.1](#): the common reference string is derived once per tag and reused across proofs.

The main distinction between our definition and prior work [BCJP24, PP26] is that we require statistical extraction or soundness: we require that a witness for instance $f(\tau)$ can be extracted from *any* accepting proof, regardless of how it was computed, which is essential for our applications. This requires building the BARG from scratch, which we do in [Appendix B](#).

Definition 4.3 (Functional Index BARG with Statistical Extraction [PP26, adapted]). Let $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of tag spaces. When λ is clear from context, we abuse notation and write $\tau \in \mathcal{T}$ to refer to $\tau \in \mathcal{T}_\lambda$. A *functional index BARG* with statistical extraction for tag space $\mathcal{T}$ is a tuple of efficient algorithms

$$\Pi_{\text{FB}} = (\text{Setup}, \text{SetupBinding}, \text{Gen}, \text{Prove}, \text{Verify}, \text{Extract})$$

with the following syntax:

- $\text{Setup}(1^\lambda, k, 1^s, 1^{s_f}) \rightarrow \text{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$ (in binary) with $1 \leq k \leq 2^\lambda$, a circuit size bound $s \in \mathbb{N}$, and a description size bound $s_f \in \mathbb{N}$, the setup algorithm outputs public parameters pp.
- $\text{SetupBinding}(1^\lambda, k, 1^s, 1^{s_f}, f) \rightarrow (\text{pp}, \text{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$ (in binary) with $1 \leq k \leq 2^\lambda$, a circuit size bound $s \in \mathbb{N}$, a description size bound $s_f \in \mathbb{N}$, and (the description of) a function $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f , the binding setup algorithm outputs public parameters pp and a trapdoor td.
- $\text{Gen}(\text{pp}, \tau) \rightarrow \text{crs}$: On input public parameters pp and a tag $\tau \in \mathcal{T}$, the generation algorithm outputs a common reference string crs. This algorithm is deterministic.
- $\text{Prove}(\text{crs}, C, (w_1, \dots, w_k)) \rightarrow \Pi$: On input a common reference string crs, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and a vector of k witnesses, the prover algorithm outputs a proof Π .
- $\text{Verify}(\text{crs}, C, \Pi) \rightarrow b'$: On input a common reference string crs, a circuit C , and a proof Π , the verification algorithm outputs a bit $b' \in \{0, 1\}$.
- $\text{Extract}(\text{td}, \Pi) \rightarrow w$: On input the trapdoor td and a proof Π , the deterministic extraction algorithm outputs a single witness $w \in \{0, 1\}^m$, where the witness length m is fixed by pp and recorded in td.

Moreover, Π_{FB} should satisfy the following properties:

- **Completeness:** For all $\lambda, k, s, s_f \in \mathbb{N}$ with $\lambda \geq 1$ and $1 \leq k \leq 2^\lambda$, all tags $\tau \in \mathcal{T}$, all circuits $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and all witnesses $w_1, \dots, w_k \in \{0, 1\}^m$ such that $C(i, w_i) = 1$ for all $i \in [k]$, we have:

$$\Pr \left[\text{Verify}(\text{crs}, C, \Pi) = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda, k, 1^s, 1^{s_f}) \\ \text{crs} = \text{Gen}(\text{pp}, \tau) \\ \Pi \leftarrow \text{Prove}(\text{crs}, C, (w_1, \dots, w_k)) \end{array} \right] = 1.$$

- **Mode indistinguishability:** For any efficient adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:

1. On input 1^λ , the adversary $\mathcal{A}$ sends $k, 1^s, 1^{s_f}$ with $1 \leq k \leq 2^\lambda$, and the description of a function $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f to the challenger.
2. The challenger computes the public parameters as follows:
 - If $b = 0$, the challenger computes $\text{pp} \leftarrow \text{Setup}(1^\lambda, k, 1^s, 1^{s_f})$.
 - If $b = 1$, the challenger computes $(\text{pp}, \text{td}) \leftarrow \text{SetupBinding}(1^\lambda, k, 1^s, 1^{s_f}, f)$.

The challenger sends pp to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

We say that Π_{FB} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Somewhere extraction:** For any (potentially unbounded) adversary $\mathcal{A}$, we define the following game:

1. On input the security parameter 1^λ , the adversary $\mathcal{A}$ sends $k, 1^s, 1^{s_f}$ with $1 \leq k \leq 2^\lambda$, and the description of a function $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f to the challenger.
2. The challenger computes $(pp, td) \leftarrow \text{SetupBinding}(1^\lambda, k, 1^s, 1^{s_f}, f)$ and sends pp to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs a tag $\tau \in \mathcal{T}$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and a proof Π .
4. The challenger computes $\text{crs} = \text{Gen}(pp, \tau)$ and extracts a witness $w \leftarrow \text{Extract}(td, \Pi)$.
5. The experiment outputs 1 if $\text{Verify}(\text{crs}, C, \Pi) = 1$ and $C(f(\tau), w) \neq 1$. Otherwise, it outputs 0.

We say that Π_{FB} satisfies somewhere extraction if for any (potentially unbounded) adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that the probability that the experiment outputs 1 is at most $\text{negl}(\lambda)$. We say that Π_{FB} satisfies *strong* somewhere extraction if the above holds even when the challenger additionally sends the trapdoor td to $\mathcal{A}$ in the second step of the game. Only the plain notion is used in this work: it is the notion that [Theorem 5.10](#) relies on, and it is the notion that [Lemma B.13](#) establishes for the recursive construction. The strong notion is recorded only because the base scheme of [Remark B.9](#) happens to satisfy it.

- **Succinctness:** There exist *efficiently computable* universal polynomials $p_{\text{crs}}(\cdot), p_\Pi(\cdot), p_{VT}(\cdot)$ that are *independent* of the tag space $\mathcal{T}$ such that for all $\lambda, k, s, s_f \in \mathbb{N}$ with $\lambda \geq 1$ and $1 \leq k \leq 2^\lambda$, all tags $\tau \in \mathcal{T}$, all $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and all witnesses $w_1, \dots, w_k \in \{0, 1\}^m$:
 - **Compact crs:** For all pp in the supports of $\text{Setup}(1^\lambda, k, 1^s, 1^{s_f})$ and $\text{SetupBinding}(1^\lambda, k, 1^s, 1^{s_f}, \cdot)$, we have $|\text{crs}| \leq p_{\text{crs}}(\lambda, \log k, s, s_f)$, where $\text{crs} = \text{Gen}(pp, \tau)$.
 - **Succinct proof:** For all pp in the support of $\text{Setup}(1^\lambda, k, 1^s, 1^{s_f})$ and all proofs Π in the support of $\text{Prove}(\text{crs}, C, (w_1, \dots, w_k))$, where $\text{crs} = \text{Gen}(pp, \tau)$, we have $|\Pi| \leq p_\Pi(\lambda, \log k, s, s_f)$.
 - **Verification time:** For all pp in the support of $\text{Setup}(1^\lambda, k, 1^s, 1^{s_f})$ and all $\Pi \in \{0, 1\}^*$, the running time of $\text{Verify}(\text{crs}, C, \Pi)$, where $\text{crs} = \text{Gen}(pp, \tau)$, is bounded by $p_{VT}(\lambda, \log k, s, s_f, |\Pi|)$.

5 BARG for Trapdoor Threshold Languages

In this section, we define and construct a non-interactive batch argument for what we refer to as *trapdoor threshold languages*. Very importantly, our construction will provide a form of statistical soundness, which we elaborate on later in this section. The first thing we do is define trapdoor threshold languages.

Trapdoors for threshold languages. Let $k \geq 1$ be an integer we refer to as the batch size. Let $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ be a Boolean circuit. For any $i \in [k]$, the i^{th} instance is a yes if it belongs to the following language:

$$\mathcal{L}_C = \{i \in [k] : \exists w \in \{0, 1\}^m \text{ s.t. } C(i, w) = 1\}$$

The threshold language is defined as follows:

$$\mathcal{L}_{\text{Th}} := \{(C, t) : |\mathcal{L}_C| \geq t\}$$

In a nutshell, a trapdoor for $\mathcal{L}_C$ can recognize (some) *no* instances (i.e., $\text{Ext}(i) = 0 \Rightarrow i \notin \mathcal{L}_C$). And moreover, for any interval $\mathcal{J} = [j_1, j_2] \subseteq [k]$, the trapdoor can *efficiently* compute the sum $\sum_{i \in \mathcal{J}} \text{Ext}(i)$, in time $O(\log k)$ instead of the naive $O(|\mathcal{J}|)$. As it turns out, it will be easier to define a binary tree whose leaves correspond to instances in $[k]$, and let Ext take as input a node and output the sum of the recognized leaves in the node's subtree.

Definition 5.1 (Trapdoor Threshold Languages). For simplicity, assume that k is a power of two, and let T be a complete binary tree with k leaves, whose leaves are labeled by the indices $1, \dots, k$ in left-to-right order and whose internal nodes are labeled by the indices $k+1, \dots, 2k-1$, by increasing height and, within each height, from left to right. More explicitly, the i^{th} node in level ℓ has label $2k - 2^{\ell+1} + i$, where the root node is on level $\ell = 0$ and the leaves are on level $\ell = \log k$. We denote the label of the root by $\text{root} = 2k-1$. Let $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ be a function mapping the nodes of T to nonnegative integers. We say that Ext is a *trapdoor for the threshold language of C* if it satisfies the following:

- **Leaf soundness:** For every leaf $i \in [k]$, we have $\text{Ext}(i) \in \{0, 1\}$, and if $\text{Ext}(i) = 0$, then $i \notin \mathcal{L}_C$.
- **Propagation:** For every internal node $v \in [k + 1, 2k - 1]$ with children v_0 and v_1 , we have $\text{Ext}(v) = \text{Ext}(v_0) + \text{Ext}(v_1)$.

Unrolling the propagation property, $\text{Ext}(v) = \sum_i \text{Ext}(i)$ for every node v , where the sum ranges over all leaves i in the subtree of T rooted at v .³ In particular, $\text{Ext}(v)$ counts the leaves flagged by Ext in the subtree of v , and by leaf soundness every yes instance in that subtree is flagged, so $\text{Ext}(v)$ upper bounds the intersection of the subtree with $\mathcal{L}_C$. Hence, whenever $\text{Ext}(\text{root}) < t$, fewer than t of the k instances are satisfiable.

Defining BARGs for trapdoor threshold languages. Our definition has two computationally indistinguishable setup modes: normal and sound. The sound mode takes as input a trapdoor Ext and guarantees that for any Boolean circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ such that Ext is a trapdoor for the threshold language of C , then with overwhelming probability, there does not exist a proof that $(C, t) \in \mathcal{L}_{\text{Th}}$ for any $t > \text{Ext}(\text{root})$.

Definition 5.2 (BARG for Trapdoor Threshold Languages). A BARG for trapdoor threshold languages is a tuple of efficient algorithms $\Pi_{\text{BARG}} = (\text{Gen}, \text{SoundGen}, \text{Prove}, \text{Verify})$ with the following syntax:

- $\text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t}) \rightarrow \text{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the batch size $k \in \mathbb{N}$ with $k \leq 2^\lambda$ (in binary), a circuit size bound $s_c \in \mathbb{N}$, and a trapdoor size bound $s_t \in \mathbb{N}$, the setup algorithm outputs a common reference string crs .
- $\text{SoundGen}(1^\lambda, k, 1^{s_c}, 1^{s_t}, \text{Ext}) \rightarrow \text{crs}$: On input the parameters $(1^\lambda, k, 1^{s_c}, 1^{s_t})$ as above and (the description of) a function $\text{Ext}: [2k - 1] \rightarrow \mathbb{N}$ on the nodes of T whose description size is at most s_t , the sound setup algorithm outputs a common reference string crs .
- $\text{Prove}(\text{crs}, C, S, (w_i)_{i \in S}) \rightarrow \Pi$: On input a common reference string crs , a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s_c , a subset $S \subseteq [k]$, and witnesses $(w_i)_{i \in S}$ with each $w_i \in \{0, 1\}^m$, the prover algorithm outputs a proof Π .
- $\text{Verify}(\text{crs}, C, t, \Pi) \rightarrow b'$: On input a common reference string crs , a circuit C , a threshold $t \in \mathbb{N}$ with $t \leq k$, and a proof Π , the verification algorithm outputs a bit $b' \in \{0, 1\}$.

Moreover, Π_{BARG} should satisfy the following properties:

- **Completeness:** For all $\lambda, k, t, s_c, s_t \in \mathbb{N}$ with $\lambda, k \geq 1$ and $t \leq k \leq 2^\lambda$, all circuits $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s_c , all subsets $S \subseteq [k]$ with $|S| \geq t$, and all witnesses $(w_i)_{i \in S}$ such that $C(i, w_i) = 1$ for all $i \in S$, we have:

$$\Pr \left[\text{Verify}(\text{crs}, C, t, \Pi) = 1 \mid \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t}) \\ \Pi \leftarrow \text{Prove}(\text{crs}, C, S, (w_i)_{i \in S}) \end{array} \right] = 1.$$

- **Mode indistinguishability:** For any efficient adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:

1. On input 1^λ , the adversary $\mathcal{A}$ sends a batch size $k \leq 2^\lambda$, $1^{s_c}, 1^{s_t}$ and the description of a function $\text{Ext}: [2k - 1] \rightarrow \mathbb{N}$ of size at most s_t to the challenger.
2. The challenger computes the common reference string:
 - If $b = 0$, it samples $\text{crs} \leftarrow \text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$.
 - If $b = 1$, it samples $\text{crs} \leftarrow \text{SoundGen}(1^\lambda, k, 1^{s_c}, 1^{s_t}, \text{Ext})$.

The challenger sends crs to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

³Note that Ext also allows computing the number of flagged leaves in any given interval $[j_1, j_2] \subseteq [k]$ by computing Ext on at most h nodes.

We say that Π_{BARG} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Statistical soundness:** For any (potentially unbounded) adversary $\mathcal{A}$, we define the following game:
 1. On input the security parameter 1^λ , the adversary $\mathcal{A}$ sends a batch size $k \leq 2^\lambda$, 1^{s_c} , 1^{s_t} and the description of a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t to the challenger.
 2. The challenger computes $\text{crs} \leftarrow \text{SoundGen}(1^\lambda, k, 1^{s_c}, 1^{s_t}, \text{Ext})$ and sends crs to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s_c , a threshold $t \in \mathbb{N}$ with $t \leq k$, and a proof Π .
 4. The experiment outputs 1 if and only if all of the following hold:
 - $\text{Verify}(\text{crs}, C, t, \Pi) = 1$.
 - Ext is a trapdoor for the batch language of C .
 - $\text{Ext}(\text{root}) < t$.

We say that Π_{BARG} satisfies statistical soundness if for any (potentially unbounded) adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that the probability that the experiment outputs 1 is at most $\text{negl}(\lambda)$.

- **Succinctness:** There exist universal polynomials $q_{\text{crs}}(\cdot)$, $q_\Pi(\cdot)$, $q_{VT}(\cdot)$ such that for all $\lambda, k, t, s_c, s_t \in \mathbb{N}$ with $\lambda, k \geq 1$ and $t \leq k \leq 2^\lambda$, all $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s_c , all subsets $S \subseteq [k]$, and all witnesses $(w_i)_{i \in S}$ with $w_i \in \{0, 1\}^m$:
 - **Succinct CRS:** For all crs in the support of $\text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$, we have $|\text{crs}| \leq q_{\text{crs}}(\lambda, \log k, s_c, s_t)$.
 - **Succinct proof:** For all crs in the support of $\text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$ and all Π in the support of $\text{Prove}(\text{crs}, C, S, (w_i)_{i \in S})$, we have $|\Pi| \leq q_\Pi(\lambda, \log k, s_c, s_t)$.
 - **Verification time:** For all crs in the support of $\text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$ and all $\Pi \in \{0, 1\}^*$, the running time of $\text{Verify}(\text{crs}, C, t, \Pi)$ is bounded by $q_{VT}(\lambda, \log k, s_c, s_t, |\Pi|)$.

5.1 Construction

In this section, we construct a BARG for trapdoor threshold languages from statistical functional BARGs and a dynamic somewhere-extractable hash.

Levels and local indexing. Throughout this section we assume that k is a power of two with $2 \leq k \leq 2^\lambda$, and we write $h := \log k$ for the height of the tree T , in particular $1 \leq h \leq \lambda$. For $\ell \in [0, h]$, the ℓ^{th} level of T consists of the 2^ℓ nodes at distance ℓ from the root, thus level 0 consists of the root alone and level h of the leaves. It is convenient to index nodes by their position within their level: for $\ell \in [0, h]$ and $i \in [2^\ell]$, we write $\text{node}_{\ell, i} \in [2k-1]$ for the i^{th} node of level ℓ in left-to-right order. Explicitly, the labeling fixed above is

$$\text{node}_{\ell, i} = 2k - 2^{\ell+1} + i,$$

since the levels $h, h-1, \dots, \ell+1$ contribute $2k - 2^{\ell+1}$ nodes in total. Hence $\text{node}_{h, i} = i$ for every leaf $i \in [k]$, we have $\text{node}_{0,1} = \text{root} = 2k-1$, and the two children of an internal node $\text{node}_{\ell, i}$ are $\text{node}_{\ell+1, 2i-1}$ (left) and $\text{node}_{\ell+1, 2i}$ (right). For each level $\ell \in [0, h]$, we write $\text{layer}_\ell = (\text{node}_{\ell, i})_{i \in [2^\ell]}$ for the vector of all nodes in level ℓ .

Construction 5.3 (BARG for Trapdoor Threshold Languages). The construction uses the following cryptographic ingredients:

- A dynamic somewhere-extractable hash $\Pi_{\text{DSEH}} = (\text{DSEH.Setup}, \text{DSEH.SetupBinding}, \text{DSEH.Gen}, \text{DSEH.Hash}, \text{DSEH.Verify}, \text{DSEH.Extract})$ as per [Definition 4.1](#), for the block space $\mathcal{X} := [2^\lambda] \times ([0, 2^\lambda] \cup [0, 2^\lambda]^3)$ of node blocks (defined in Prove below) and the tag space $\mathcal{T} := \{0, 1\}^{p_{\text{hk}}} \cup \{\perp\}$, where $p_{\text{hk}} := p_{\text{hk}}(\lambda, \log k, s_d, \log s_f)$ is the bound on the hash key size from [Definition 4.1](#) when instantiated with the parameters we define below. Note that there is no circular dependence since p_{hk} is required to be independent of the tag space.
- A functional index BARG $\Pi_{\text{FB}} = (\text{FB.Setup}, \text{FB.SetupBinding}, \text{FB.Gen}, \text{FB.Prove}, \text{FB.Verify}, \text{FB.Extract})$ with somewhere extraction as per [Definition 4.3](#), for the tag space $\mathcal{T} := \{0, 1\}^{p_{\text{hk}}} \cup \{\perp\}$, for the same p_{hk} from the previous bullet.

We define $\text{ComputeParams}: \mathbb{N}^4 \rightarrow \mathbb{N}^2$ to be a function that takes a security parameter λ , a level ℓ and parameters $s_d^{(\ell-1)}, s_f^{(\ell-1)}$ for the previous level, and outputs parameters $s_d^{(\ell)}, s_f^{(\ell)}$ for the current level. The function is computed as follows:

$$\begin{aligned} & \text{ComputeParams}(\lambda, \ell, s_d^{(\ell-1)}, s_f^{(\ell-1)}) \\ &= \left(s_f^{(\ell)} = s_t + p_{\text{Extract}}(\lambda, \ell, s_d^{(\ell-1)}, \log s_f^{(\ell-1)}) + 1000 \log k, s_d^{(\ell)} = s_t + d_{\text{Extract}}(\lambda, \ell, \log s_d^{(\ell-1)}, \log s_f^{(\ell-1)}) + 1000 \right) \end{aligned}$$

where p_{Extract} and d_{Extract} are the polynomials from [Definition 4.1](#).

We will argue later on that this function is efficiently computable and outputs legal parameters that can be used in our construction. Let s_D bound the size of the circuits $D^{(\ell)}$ of [Figure 2](#) and [Figure 3](#). We assume $k \geq 2$ is a power of 2, so $h \geq 1$ is an integer. The algorithms of the scheme are defined as follows:

- $\text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$: On input the security parameter $\lambda \in \mathbb{N}$, the batch size $k \in \mathbb{N}$, a circuit size bound $s_c \in \mathbb{N}$, and a trapdoor size bound $s_t \in \mathbb{N}$, the setup algorithm proceeds as follows:
 1. Set $s_f^{(-1)} = 1$ and $s_d^{(-1)} = 1$.
 2. Let $h := \log k$. For each level $\ell = 0, \dots, h$, starting from the root level 0 down to the leaves level h :
 - Compute $s_d^{(\ell)}, s_f^{(\ell)} = \text{ComputeParams}(\lambda, \ell, s_d^{(\ell-1)}, s_f^{(\ell-1)})$.
 - Sample the hash parameters $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$.
 - Sample the functional BARG public parameters $\text{pp}_\ell^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$.
 3. Output $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$.
- $\text{SoundGen}(1^\lambda, k, 1^{s_c}, 1^{s_t}, \text{Ext})$: On input the security parameter $\lambda \in \mathbb{N}$, the batch size $k \in \mathbb{N}$, a circuit size bound $s_c \in \mathbb{N}$, a trapdoor size bound $s_t \in \mathbb{N}$, and (the description of) a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t , the sound setup algorithm proceeds as follows:
 1. Set $s_f^{(0)} = 1$ and $s_d^{(0)} = 1$.
 2. Let $h := \log k$. For each level $\ell = 0, \dots, h$, starting from the root level 0 down to the leaves level h :
 - Compute $s_d^{(\ell)}, s_f^{(\ell)} = \text{ComputeParams}(\lambda, \ell, s_d^{(\ell-1)}, s_f^{(\ell-1)})$.
 - Sample binding hash parameters and a trapdoor $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$, for the function g_ℓ of [Figure 1](#). This is a legal step by [Proposition 5.4](#).
 - Sample binding functional BARG public parameters and a trapdoor $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$, with the same function g_ℓ .
 3. Output $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$.

Here $g_0 \equiv 1$, and for $\ell > 0$ the function g_ℓ is the following.

Constants: The trapdoor function Ext and the hash trapdoor $\text{td}_{\ell-1}^{\text{DSEH}}$ of the level above.
Inputs: A tag $\tau_\ell = \text{dig}_{\ell-1}$ consisting of the digest of the level $\ell - 1$.

1. Extract the block committed at the index on which the hash of level $\ell - 1$ is bound,

$$x' = \text{DSEH.Extract}(\text{td}_{\ell-1}^{\text{DSEH}}, \text{dig}_{\ell-1}).$$

2. If x' is not of the form $(i, v'_{\text{node}_{\ell-1,i}}, v'_{\text{node}_{\ell,2i-1}}, v'_{\text{node}_{\ell,2i}})$ with $i \in [2^{\ell-1}]$, output the index 1.
3. Otherwise parse x' in this form, so that x' is the block of the node $\text{node}_{\ell-1,i}$, and compute the bit

$$\beta = \begin{cases} 0 & v'_{\text{node}_{\ell,2i}} > \text{Ext}(\text{node}_{\ell,2i}) \\ 1 & \text{otherwise} \end{cases}$$

4. Output the index $2i - \beta$.

Figure 1: Function $g_\ell[\text{Ext}, \text{td}_{\ell-1}^{\text{DSEH}}]$ for a level $\ell > 0$.

- $\text{Prove}(\text{crs}, C, S, (w_i)_{i \in S})$: On input the common reference string $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$, a subset $S \subseteq [k]$, and witnesses $(w_i)_{i \in S}$ with each $w_i \in \{0, 1\}^m$, the prover algorithm proceeds as follows:

1. For each $i = 1, \dots, 2k - 1$ (starting from 1 and ending at the root $2k - 1$), compute the node values of the tree T as follows:

- For each leaf $i \in [k]$, set $v_i = \begin{cases} 1 & i \in S \\ 0 & i \notin S \end{cases}$
- For each internal node $i \in [k + 1, 2k - 1]$ with children $L, R \in [i - 1]$, set $v_i = v_L + v_R$.

Recall that for each layer ℓ and node index $i \in [2^\ell]$, the value $\text{node}_{\ell,i}$ is the global index of the i^{th} node in layer ℓ .

2. For each level $\ell \in [0, h]$ and each $i \in [2^\ell]$ set

$$x_{\ell,i} := \begin{cases} (i, v_{\text{node}_{\ell,i}}, v_{\text{node}_{\ell+1,2i-1}}, v_{\text{node}_{\ell+1,2i}}) & \ell \in [0, h - 1] \\ (i, v_{\text{node}_{h,i}}) & \ell = h \end{cases}$$

Denote $\mathbf{x}_\ell := (x_{\ell,1}, \dots, x_{\ell,2^\ell})$ for the vector of blocks of level ℓ .

3. Set $\tau_0 = \perp$, then for each level $\ell = 0, \dots, h$:
 - Compute the hash key $\text{hk}_\ell = \text{DSEH.Gen}(\text{pp}_\ell^{\text{DSEH}}, \tau_\ell)$ and the functional BARG common reference string $\text{crs}_\ell^{\text{FB}} = \text{FB.Gen}(\text{pp}_\ell^{\text{FB}}, \tau_\ell)$.
 - Hash the blocks of the level $(\text{dig}_\ell, \sigma_{\ell,1}, \dots, \sigma_{\ell,2^\ell}) = \text{DSEH.Hash}(\text{hk}_\ell, \mathbf{x}_\ell)$.
 - If $\ell < h$ then set the tag for the next level $\tau_{\ell+1} = \text{dig}_\ell$.
4. Compute a functional BARG proof for every level:
 - For each level $\ell \in [0, h - 1]$ and each $i \in [2^\ell]$, set the witness

$$\omega_{\ell,i} = (x_{\ell,i}, \sigma_{\ell,i}, x_{\ell+1,2i-1}, \sigma_{\ell+1,2i-1}, x_{\ell+1,2i}, \sigma_{\ell+1,2i}),$$

and compute

$$\Pi_\ell \leftarrow \text{FB.Prove}(\text{crs}_\ell^{\text{FB}}, D^{(\ell)}[v_{\text{root}}, \text{hk}_\ell, \text{hk}_{\ell+1}, \text{dig}_\ell, \text{dig}_{\ell+1}], (\omega_{\ell,i})_{i \in [2^\ell]}),$$

where $D^{(\ell)}$ is the node circuit of [Figure 3](#).

- For the leaves level $\ell = h$, for each leaf $j \in [k]$ set the witness $\omega_{h,j} = (x_{h,j}, \sigma_{h,j}, w_j)$, where w_j is the witness given as input if $j \in S$ and $w_j = 0^m$ otherwise, and compute

$$\Pi_h \leftarrow \text{FB.Prove}(\text{crs}_h^{\text{FB}}, D^{(h)}[C, \text{hk}_h, \text{dig}_h], (\omega_{h,j})_{j \in [k]}),$$

where $D^{(h)}$ is the leaves circuit of Figure 2.

5. Output the proof $\Pi = (v_{\text{root}}, \text{dig}_0, \dots, \text{dig}_h, \Pi_0, \dots, \Pi_h)$.

We note that there is a redundancy in the values hashed at each level, since each value is hashed as a parent and also as a child. This becomes important in the proof as the digest encodes a node that is violating (if any) in the *next* node.

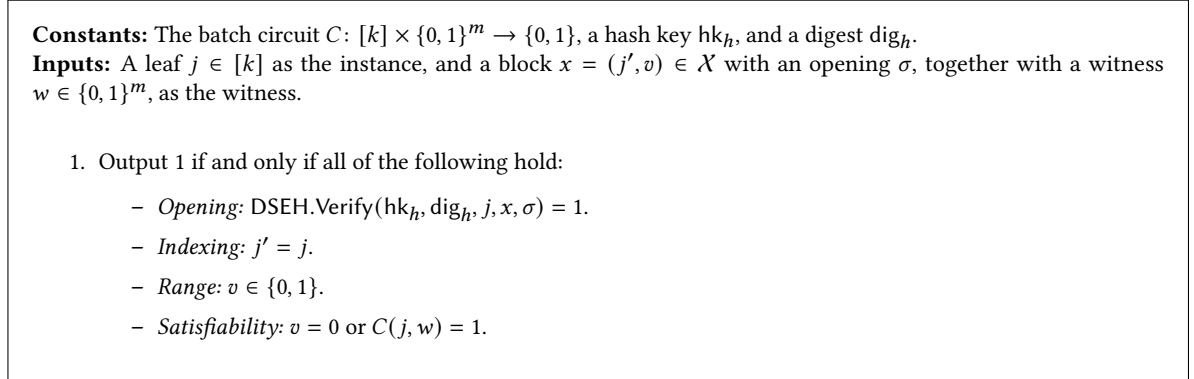


Figure 2: Circuit $D^{(h)}[C, \text{hk}_h, \text{dig}_h]$ for the leaves level.

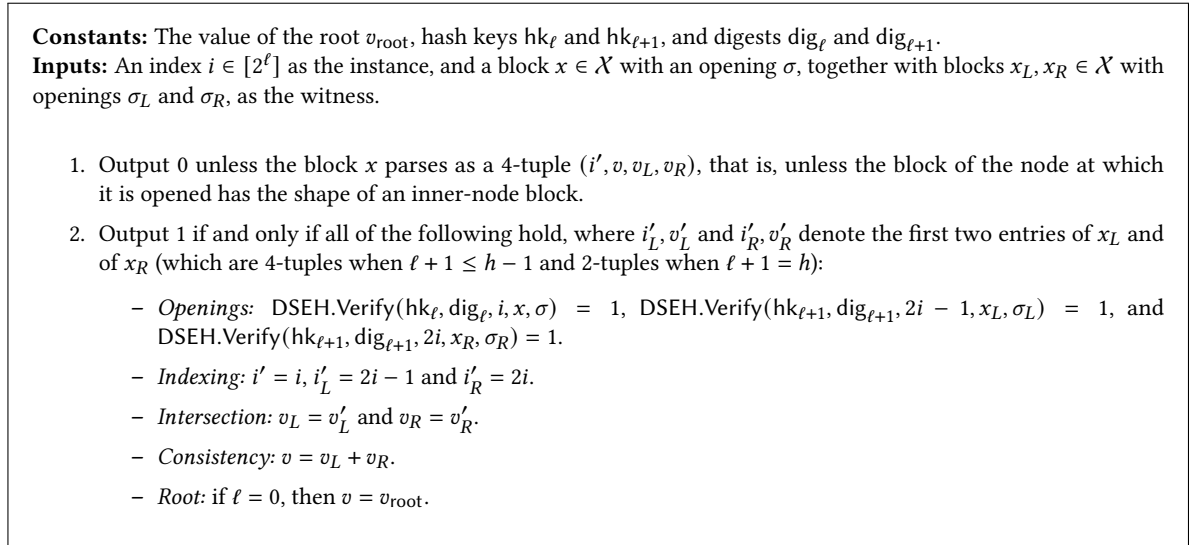


Figure 3: Circuit $D^{(\ell)}[v_{\text{root}}, \text{hk}_\ell, \text{hk}_{\ell+1}, \text{dig}_\ell, \text{dig}_{\ell+1}]$ for a level $\ell \in [0, h - 1]$.

- $\text{Verify}(\text{crs}, C, t, \Pi)$: On input the common reference string $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$, a threshold $t \in \mathbb{N}$, and a proof $\Pi = (v_{\text{root}}, \text{dig}_0, \dots, \text{dig}_h, \Pi_0, \dots, \Pi_h)$, the verification algorithm proceeds as follows:

1. Output 0 if the proof Π is not of this format.

2. Recompute the tags, the hash keys and the functional BARG common reference strings as Prove does: set $\tau_0 = \perp$ and, for each level $\ell = 0, \dots, h$, compute $\text{hk}_\ell = \text{DSEH.Gen}(\text{pp}_\ell^{\text{DSEH}}, \tau_\ell)$ and $\text{crs}_\ell^{\text{FB}} = \text{FB.Gen}(\text{pp}_\ell^{\text{FB}}, \tau_\ell)$ and, if $\ell < h$, set $\tau_{\ell+1} = \text{dig}_\ell$.
3. Output 1 if and only if all of the following hold:
 - *Threshold*: $v_{\text{root}} \geq t$.
 - *Inner levels*: $\text{FB.Verify}(\text{crs}_\ell^{\text{FB}}, D^{(\ell)}[v_{\text{root}}, \text{hk}_\ell, \text{hk}_{\ell+1}, \text{dig}_\ell, \text{dig}_{\ell+1}], \Pi_\ell) = 1$ for every level $\ell \in [0, h-1]$, where $D^{(\ell)}$ is the circuit of Figure 3.
 - *Leaves level*: $\text{FB.Verify}(\text{crs}_h^{\text{FB}}, D^{(h)}[C, \text{hk}_h, \text{dig}_h], \Pi_h) = 1$, where $D^{(h)}$ is the circuit of Figure 2.

We now show that Construction 5.3 is complete and succinct.

Proposition 5.4 (Legality of the Bounds). *For every level $\ell \in [h]$, the circuit g_ℓ of Figure 1 has size at most $s_f^{(\ell)}$ and depth at most $s_d^{(\ell)}$.*

Proof. The circuit g_ℓ hardcodes Ext and $\text{td}_{\ell-1}^{\text{DSEH}}$, and evaluates Ext , then DSEH.Extract , then one shape check, one comparison and one subtraction. Since a circuit's depth is at most its size, Ext costs size and depth at most s_t . By the succinct trapdoor property of Definition 4.1 with the bounds $s_f^{(\ell-1)}, s_d^{(\ell-1)}$ of level $\ell-1$, running DSEH.Extract costs size at most $p_{\text{Extract}}^{(\ell)}$ and depth at most $d_{\text{Extract}}^{(\ell)}$. The three remaining steps act on the index $i \in [2^{\ell-1}]$, on the three values recorded in x' , which lie in $[0, k]$, and on $\text{Ext}(\text{node}_{\ell,2i}) \in [0, k]$, so on integers of at most $\lceil \log k \rceil + 2$ bits. They therefore take at most $100 \log k$ gates (and $100 \log k$ layers). $\square$

Lemma 5.5 (Completeness). *Suppose that Π_{DSEH} is correct (Definition 4.1) and that Π_{FB} is complete (Definition 4.3). Then Construction 5.3 is complete (Definition 5.2).*

Proof. Fix $\lambda, k, t, s_c, s_t \in \mathbb{N}$ with $\lambda, k \geq 1$ and $t \leq k \leq 2^\lambda$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s_c , a subset $S \subseteq [k]$ with $|S| \geq t$, and witnesses $(w_i)_{i \in S}$ with $C(i, w_i) = 1$ for all $i \in S$. Let $\text{crs} \leftarrow \text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$ and $\Pi \leftarrow \text{Prove}(\text{crs}, C, S, (w_i)_{i \in S})$, and let $\{v_i\}, \{x_{\ell,i}\}, \{\text{hk}_\ell, \text{crs}_\ell^{\text{FB}}, \text{dig}_\ell, \sigma_{\ell,i}\}$ and $\{\tau_\ell\}$ be the values, blocks, keys, hashes and tags computed by the prover. Since Verify recomputes the tags, the hash keys and the functional BARG common reference strings by the same rule as Prove and from the same digests, and since DSEH.Gen and FB.Gen are deterministic (Definition 4.1 and Definition 4.3), the two agree on every $\tau_\ell, \text{hk}_\ell$ and $\text{crs}_\ell^{\text{FB}}$, and hence on the $h+1$ circuits. It therefore suffices to argue that the threshold check passes and that each of the $h+1$ functional BARG proofs verifies.

The threshold check passes. We claim that $v_{\text{node}_{\ell,i}}$ counts the leaves of S in the subtree rooted at $\text{node}_{\ell,i}$, by induction from the leaves upwards: for a leaf, $v_{\text{node}_{h,i}} = 1$ exactly when $i \in S$, and an internal node receives the sum of the values of its two children. In particular $v_{\text{root}} = v_{\text{node}_{0,1}} = |S| \geq t$.

Every instance of every level is satisfied. By completeness of Π_{FB} , applied to the parameters $\text{pp}_\ell^{\text{FB}}$ in the support of $\text{FB.Setup}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$ and the string $\text{crs}_\ell^{\text{FB}} = \text{FB.Gen}(\text{pp}_\ell^{\text{FB}}, \tau_\ell)$, it suffices to check that the witness the prover supplies for each instance satisfies the corresponding circuit. Recall that the level- ℓ circuit has size at most s_D and that the level- ℓ instance of Π_{FB} is set up with the batch size 2^ℓ , which is the number of instances proved at that level. Throughout we use correctness of Π_{DSEH} : for every level ℓ , the digest and the openings satisfy $(\text{dig}_\ell, \sigma_{\ell,1}, \dots, \sigma_{\ell,2^\ell}) = \text{DSEH.Hash}(\text{hk}_\ell, \mathbf{x}_\ell)$ with $\text{hk}_\ell = \text{DSEH.Gen}(\text{pp}_\ell^{\text{DSEH}}, \tau_\ell)$ and $\text{pp}_\ell^{\text{DSEH}}$ in the support of $\text{DSEH.Setup}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$, which are legal parameters by Proposition 5.4, so $\text{DSEH.Verify}(\text{hk}_\ell, \text{dig}_\ell, i, x_{\ell,i}, \sigma_{\ell,i}) = 1$ for every $i \in [2^\ell]$.

- *Level $\ell \in [0, h-1]$.* Fix $i \in [2^\ell]$ and the witness $\omega_{\ell,i} = (x_{\ell,i}, \sigma_{\ell,i}, x_{\ell+1,2i-1}, \sigma_{\ell+1,2i-1}, x_{\ell+1,2i}, \sigma_{\ell+1,2i})$ of Figure 3. The three opening checks hold by the previous paragraph, applied to levels ℓ and $\ell+1$, and the indexing checks hold because $x_{\ell,i}, x_{\ell+1,2i-1}$ and $x_{\ell+1,2i}$ record the indices $i, 2i-1$ and $2i$. For the intersection check, the values that $x_{\ell,i}$ records for the two children of $\text{node}_{\ell,i}$ are $v_{\text{node}_{\ell+1,2i-1}}$ and $v_{\text{node}_{\ell+1,2i}}$, which are exactly the values that $x_{\ell+1,2i-1}$ and $x_{\ell+1,2i}$ record for themselves. The consistency check holds because $v_{\text{node}_{\ell,i}} = v_{\text{node}_{\ell+1,2i-1}} + v_{\text{node}_{\ell+1,2i}}$, by the way the prover assigns values to internal nodes. Finally, if $\ell = 0$ then $i = 1$ and the root check holds because $\text{node}_{0,1} = \text{root}$.

- *Leaves level.* Fix a leaf $j \in [k]$ and the witness $\omega_{h,j} = (x_{h,j}, \sigma_{h,j}, w_j)$ of Figure 2. The opening check holds by the previous paragraph, and the indexing check holds because $x_{h,j} = (j, v_{\text{node}_{h,j}})$ records the index j . The range check holds because $v_{\text{node}_{h,j}} \in \{0, 1\}$ by definition. For the satisfiability check, if $v_{\text{node}_{h,j}} = 1$ then $j \in S$, so w_j is the witness given to the prover and $C(j, w_j) = 1$.

Hence every instance of every level is satisfied, so by completeness of Π_{FB} all $h+1$ proofs verify, and $\text{Verify}(\text{crs}, C, t, \Pi) = 1$ with probability 1. $\square$

Lemma 5.6 (Succinctness). *Suppose that Π_{DSEH} and Π_{FB} are succinct (Definition 4.1 and Definition 4.3) and that both run in polynomial time. Then Construction 5.3 is succinct (Definition 5.2).*

Proof. Let $p_{\text{hk}}, p_{\text{dig}}, p_{\sigma}, p_{\text{Extract}}, d_{\text{Extract}}$ be the polynomials of Definition 4.1 and let $p_{\text{crs}}, p_{\Pi}, p_{VT}$ be those of Definition 4.3. First, we bound the parameters $s_f^{(\ell)}, s_d^{(\ell)}$ and s_D consumed by the building blocks:

- **Description size and depth of the functions $g_{\ell}(s_f^{(\ell)})$ and $s_d^{(\ell)}$:** The level- ℓ bounds depend on those of level $\ell - 1$ through p_{Extract} and d_{Extract} , hence on $s_f^{(\ell-1)}$ only polylogarithmically and on $s_d^{(\ell-1)}$ polynomially only in $s_f^{(\ell)}$. Both recurrences therefore admit a single bound for all the levels at once, namely a polynomial q_{par} with

$$s_f^{(\ell)}, s_d^{(\ell)} \leq q_{\text{par}}(\lambda, \log k, s_t) \quad \text{for every } \ell \in [0, h].$$

- **Size of $D^{(\ell)}(s_D)$:** The circuit in Figure 3 is dominated by verifying openings of Π_{DSEH} and a couple of consistency checks, and since DSEH.Verify runs in polynomial time, the circuit size is polynomial in $p_{\text{hk}}, p_{\text{dig}}$ and p_{σ} , all evaluated at $(\lambda, \log k, s_d^{(\ell)}, \log s_f^{(\ell)})$. The circuit in Figure 2 additionally evaluates C , of size at most s_c . With the previous bullet, there exists a polynomial q_s such that $s_D \leq q_s(\lambda, \log k, s_c, s_t)$.

Every bound below is one of the polynomials above evaluated at $s_f^{(\ell)}, s_d^{(\ell)}$ and s_D , summed over the $h+1 = \log k + 1$ levels of the tree, so substituting the two bullets leaves a polynomial in $\lambda, \log k, s_c$ and s_t .

Compact CRS. A common reference string consists of the $h+1$ hash parameters and the $h+1$ functional BARG public parameters. Since DSEH.Setup runs in polynomial time on input $(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$, with the number of blocks given in binary, $|\text{pp}_{\ell}^{\text{DSEH}}|$ is polynomial in $\lambda, \log k, s_f^{(\ell)}$ and $s_d^{(\ell)}$. Likewise FB.Setup runs in polynomial time on input $(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$, with the number of instances given in binary, so $|\text{pp}_{\ell}^{\text{FB}}|$ is polynomial in $\lambda, \log k, s_D$ and $s_f^{(\ell)}$. The same holds for FB.SetupBinding , whose additional input is a description of size at most $s_f^{(\ell)}$. Summing over the levels bounds $|\text{crs}|$ by a polynomial in $\lambda, \log k, s_c$ and s_t , and we take q_{crs} to be it.

Succinct proof. A proof consists of the root value, the $h+1$ digests and the $h+1$ functional BARG proofs. The root value satisfies $v_{\text{root}} \in [0, k]$ and is described by $O(\log k)$ bits. By the succinct digest property of Π_{DSEH} , we have $|\text{dig}_{\ell}| \leq p_{\text{dig}}(\lambda, \log k, s_d^{(\ell)}, \log s_f^{(\ell)})$, and by the succinct proof property of Π_{FB} , we have $|\Pi_{\ell}| \leq p_{\Pi}(\lambda, \log k, s_D, s_f^{(\ell)})$. Summing over the levels bounds $|\Pi|$ by a polynomial in $\lambda, \log k, s_c$ and s_t , and we take q_{Π} to be it.

Verification time. The verifier parses crs and Π , which takes time $O(|\text{crs}| + |\Pi|)$. It then recomputes the $h+1$ tags, hash keys and functional BARG common reference strings: each tag is a digest carried by Π , and DSEH.Gen and FB.Gen run in polynomial time in the length of their inputs, so each hash key and each $\text{crs}_{\ell}^{\text{FB}}$ is computed in time polynomial in $\lambda, \log k, s_f^{(\ell)}, s_d^{(\ell)}, s_D$ and $|\Pi|$, in particular $|\text{crs}_{\ell}^{\text{FB}}| \leq p_{\text{crs}}(\lambda, \log k, s_D, s_f^{(\ell)})$ by the compact CRS property of Π_{FB} . It assembles the $h+1$ circuits, each of size at most s_D , and runs FB.Verify once per level. By the verification time property of Π_{FB} , each of these runs in time $p_{VT}(\lambda, \log k, s_D, s_f^{(\ell)}, |\Pi_{\ell}|)$, and $|\Pi_{\ell}| \leq |\Pi|$. The threshold check compares two integers in $[0, k]$. Summing over the levels bounds the total running time by a polynomial in $\lambda, \log k, s_c, s_t$ and $|\Pi|$, and we take q_{VT} to be it. $\square$

Theorem 5.7 (Mode Indistinguishability). *Assume that Π_{DSEH} satisfies mode indistinguishability (Definition 4.1) and that Π_{FB} satisfies mode indistinguishability (Definition 4.3). Then Construction 5.3 satisfies mode indistinguishability (Definition 5.2).*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game of [Definition 5.2](#) against [Construction 5.3](#). We first fix its first message. The advantage of $\mathcal{A}$ is the average, over its first message, of its advantage conditioned on that message, so for every λ there is a first message $k, 1^{s_c}, 1^{s_t}$ together with the description of a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t that maximizes it. It suffices to bound the advantage of the adversary that sends this message and then runs the second stage of $\mathcal{A}$, so we assume from here on that the first message is fixed and write $h = \log k$. Since $\mathcal{A}$ is efficient and writes k in binary, $h \leq \text{poly}(\lambda)$. The point of fixing the first message is that h is now a fixed function of λ and not a random variable, so a reduction may sample a level $j \xleftarrow{R} [0, h]$ before it runs $\mathcal{A}$. All the experiments below differ only in how the challenger samples the common reference string $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ that it sends to $\mathcal{A}$. In each of them the output of the experiment is the guess b' that $\mathcal{A}$ outputs, and the trapdoors are never given to $\mathcal{A}$. The goal is to prove that the adversary cannot distinguish between the two following experiments except with negligible probability:

- $\text{Real}^{(0)}$: the mode indistinguishability game with $b = 0$, in which every level is in normal mode. Formally:
 1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^{s_c}, 1^{s_t}$ and the description of a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t .
 2. The challenger runs $\text{Gen}(1^\lambda, k, 1^{s_c}, 1^{s_t})$. That is, for each level $\ell = 0, \dots, h$:
 - Sample the hash parameters $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$.
 - Sample the functional BARG public parameters $\text{pp}_\ell^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$.
 The challenger sends $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.
- $\text{Real}^{(1)}$: the mode indistinguishability game with $b = 1$, in which every level is in binding mode. Formally:
 1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^{s_c}, 1^{s_t}$ and the description of a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t .
 2. The challenger runs $\text{SoundGen}(1^\lambda, k, 1^{s_c}, 1^{s_t}, \text{Ext})$. That is, for each level $\ell = 0, \dots, h$:
 - Sample the hash parameters $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$, where g_ℓ is defined in [Figure 1](#).
 - Sample binding functional BARG public parameters and a trapdoor $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$, with the same function g_ℓ .
 The challenger sends $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.

To do so, we define the following sequence of hybrids for each $j \in [0, h+1]$:

- Hyb_j : in this experiment, the hash and BARG parameters at levels $\ell < j$ are sampled in binding mode, whereas at levels $\ell \geq j$ they are in normal mode. Formally:
 1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^{s_c}, 1^{s_t}$ and the description of a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t .
 2. The challenger samples the following. For each level $\ell = 0, \dots, h$, if $\ell < j$:
 - Sample the hash parameters $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$, where g_ℓ is defined in [Figure 1](#).
 - Sample binding functional BARG public parameters and a trapdoor $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$, with the same function g_ℓ .
 If $\ell \geq j$:
 - Sample the hash parameters $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$.

- Sample the functional BARG public parameters $\text{pp}_\ell^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$.
- The challenger sends $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$.
- 3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.

In particular, Hyb_0 is identical to $\text{Real}^{(0)}$ and Hyb_{h+1} is identical to $\text{Real}^{(1)}$. The function g_ℓ of Figure 1 hardwires the hash trapdoor $\text{td}_{\ell-1}^{\text{DSEH}}$ of the level above it, so it is well defined in Hyb_j for every level $\ell < j$ that is in binding mode. We also define an intermediate hybrid for each $j \in [0, h]$:

- Hyb'_j : the same as Hyb_j , except that for level j the hash is sampled in binding mode, while the BARG is still in normal mode. Formally:

2. For level j , the challenger samples the following:

- Sample the hash parameters $(\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^j, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_j)$, where g_j is defined in Figure 1.
- Sample the functional BARG public parameters $\text{pp}_j^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^j, 1^{s_D}, 1^{s_f^{(j)}})$.

The rest of the experiment remains unchanged.

The chain of experiments is

$$\text{Hyb}_0, \text{Hyb}'_0, \text{Hyb}_1, \text{Hyb}'_1, \dots, \text{Hyb}_h, \text{Hyb}'_h, \text{Hyb}_{h+1},$$

where a step from Hyb_j to Hyb'_j switches the hash of level j and a step from Hyb'_j to Hyb_{j+1} switches the functional BARG of that level. We bound the two kinds of steps separately. In both claims we bound the *signed* sum over the levels rather than each level separately: this is what a single reduction that samples the level can bound, whereas bounding each level separately would again require one negligible function per level, and the number of levels grows with λ .

Claim 5.8. Assume that Π_{DSEH} satisfies mode indistinguishability. Then there exists a negligible function $\text{negl}^{\text{DSEH}}$ such that

$$\left| \sum_{j \in [0, h]} \left(\Pr[\text{Hyb}_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}'_j(\mathcal{A}) = 1] \right) \right| \leq (h+1) \cdot \text{negl}^{\text{DSEH}}(\lambda).$$

Proof. Consider the following efficient adversary $\mathcal{B}$ for the mode indistinguishability game of Π_{DSEH} (Definition 4.1), which takes a level $j \in [0, h]$ as an input.

1. Run $\mathcal{A}(1^\lambda)$ to obtain $k, 1^{s_c}, 1^{s_t}$ and the description of Ext , and set $h = \log k$.
2. For each $\ell = 0, \dots, j-1$, sample $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$ and $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$, in this order, so that the function g_ℓ of Figure 1 has the trapdoor $\text{td}_{\ell-1}^{\text{DSEH}}$ of the level above it available.
3. Send $2^j, 1^{s_f^{(j)}}, 1^{s_d^{(j)}}$ and the description of g_j to the challenger, and let $\text{pp}_j^{\text{DSEH}}$ be the public parameters it returns. The function g_j is well defined: for $j = 0$ it is the constant $g_0 \equiv 1$, and for $j > 0$ the trapdoor $\text{td}_{j-1}^{\text{DSEH}}$ was sampled in the previous step.
4. Sample $\text{pp}_j^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^j, 1^{s_D}, 1^{s_f^{(j)}})$.
5. For each level $\ell = j+1, \dots, h$, sample $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$ and $\text{pp}_\ell^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$.
6. Send $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

The adversary $\mathcal{B}$ is efficient: it runs $\mathcal{A}$ once and samples $2(h+1) \leq 2\lambda + 2$ further keys. It is admissible because g_j has description size at most $s_f^{(j)}$ and depth at most $s_d^{(j)}$, which are the bounds it sends to the challenger.

The simulation is perfect in both modes. Every level $\ell < j$ is sampled exactly as in sound mode and every level $\ell > j$ exactly as in normal mode, in both cases as the challengers of Hyb_j and of Hyb'_j do, and the functional BARG of level j is sampled in normal mode, as in both experiments. The only difference is the hash of level j : if the mode indistinguishability challenger is in mode $b = 0$ then $\text{pp}_j^{\text{DSEH}}$ is distributed as $\text{DSEH.Setup}(1^\lambda, 2^j, 1^{s_f^{(j)}}, 1^{s_d^{(j)}})$ and $\mathcal{B}(j)$ perfectly simulates Hyb_j , and if it is in mode $b = 1$ then $\text{pp}_j^{\text{DSEH}}$ is distributed as the first output of $\text{DSEH.SetupBinding}(1^\lambda, 2^j, 1^{s_f^{(j)}}, 1^{s_d^{(j)}}, g_j)$ and $\mathcal{B}(j)$ perfectly simulates Hyb'_j . Note that $\mathcal{B}$ never uses $\text{td}_j^{\text{DSEH}}$, which the challenger does not give it, and which the two experiments do not use either. Consequently, for every level j ,

$$\Pr[\mathcal{B}(j) = 1 : b = 0] - \Pr[\mathcal{B}(j) = 1 : b = 1] = \Pr[\text{Hyb}_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}'_j(\mathcal{A}) = 1].$$

Let $\mathcal{B}^*$ be the adversary that samples $j \xleftarrow{\mathcal{R}} [0, h]$ and runs $\mathcal{B}(j)$. This is well defined because h was fixed at the start of the proof. It is a single efficient adversary, so by mode indistinguishability of Π_{DSEH} there is a negligible function $\text{negl}^{\text{DSEH}}$ with

$$|\Pr[\mathcal{B}^* = 1 : b = 0] - \Pr[\mathcal{B}^* = 1 : b = 1]| \leq \text{negl}^{\text{DSEH}}(\lambda).$$

Since j is sampled uniformly and independently of the challenger's bit, the left-hand side equals

$$\left| \frac{1}{h+1} \sum_{j \in [0, h]} \left(\Pr[\mathcal{B}(j) = 1 : b = 0] - \Pr[\mathcal{B}(j) = 1 : b = 1] \right) \right| = \frac{1}{h+1} \cdot \left| \sum_{j \in [0, h]} \left(\Pr[\text{Hyb}_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}'_j(\mathcal{A}) = 1] \right) \right|,$$

and the claim follows by multiplying by $h+1$. $\square$

Claim 5.9. *There exists a negligible function negl^{FB} such that*

$$\left| \sum_{j \in [0, h]} \left(\Pr[\text{Hyb}'_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{j+1}(\mathcal{A}) = 1] \right) \right| \leq (h+1) \cdot \text{negl}^{\text{FB}}(\lambda).$$

Proof. Consider the following efficient adversary $\mathcal{C}$ for the mode indistinguishability game of Π_{FB} (Definition 4.3), which takes a level $j \in [0, h]$ as an input.

1. Run $\mathcal{A}(1^\lambda)$ to obtain $k, 1^{s_c}, 1^{s_t}$ and the description of Ext , and set $h = \log k$.
2. For each $\ell = 0, \dots, j-1$, sample $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$ and $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$, in this order.
3. Sample $(\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^j, 1^{s_f^{(j)}}, 1^{s_d^{(j)}}, g_j)$, which is the mode of the hash of level j in both experiments.
4. Send $2^j, 1^{s_D}, 1^{s_f^{(j)}}$ and the description of g_j to the challenger, and let pp_j^{FB} be the public parameters it returns.
5. For each level $\ell = j+1, \dots, h$, sample $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}})$ and $\text{pp}_\ell^{\text{FB}} \leftarrow \text{FB.Setup}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}})$.
6. Send $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

As before, $\mathcal{C}$ is efficient and admissible, and the simulation is perfect in both modes: all the levels other than j are sampled exactly as in the two experiments, the hash of level j is binding in both of them, and the only difference is the functional BARG of level j . If the challenger is in mode $b = 0$ then pp_j^{FB} is distributed as $\text{FB.Setup}(1^\lambda, 2^j, 1^{s_D},$

$1^{s_f^{(j)}}$) and $C(j)$ perfectly simulates Hyb'_j , and if it is in mode $b = 1$ then pp_j^{FB} is distributed as the first output of $\text{FB.SetupBinding}(1^\lambda, 2^j, 1^{s_D}, 1^{s_f^{(j)}}, g_j)$ and $C(j)$ perfectly simulates Hyb_{j+1} . Recall that in Hyb_{j+1} every level $\ell < j + 1$ is in binding mode, which is the case here, and every level $\ell \geq j + 1$ is in normal mode, which is the case by the fifth step. Hence, for every level j ,

$$\Pr[C(j) = 1 : b = 0] - \Pr[C(j) = 1 : b = 1] = \Pr[\text{Hyb}'_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{j+1}(\mathcal{A}) = 1],$$

and the claim follows exactly as in [Claim 5.8](#), by applying mode indistinguishability of Π_{FB} to the adversary C^* that samples $j \xleftarrow{R} [0, h]$ and runs $C(j)$, which is again well defined because h is fixed. $\square$

Combining [Claim 5.8](#) and [Claim 5.9](#) with the triangle inequality along the chain,

$$\begin{aligned} |\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{h+1}(\mathcal{A}) = 1]| &\leq (h+1) \cdot (\text{negl}^{\text{DSEH}}(\lambda) + \text{negl}^{\text{FB}}(\lambda)) \\ &\leq (\lambda+1) \cdot (\text{negl}^{\text{DSEH}}(\lambda) + \text{negl}^{\text{FB}}(\lambda)), \end{aligned}$$

using $h \leq \lambda$, which holds since $k \leq 2^\lambda$. The right-hand side is negligible, and since Hyb_0 is $\text{Real}^{(0)}$ and Hyb_{h+1} is $\text{Real}^{(1)}$, this is exactly mode indistinguishability of [Construction 5.3](#). $\square$

Theorem 5.10 (Statistical Soundness). *Suppose that Π_{DSEH} satisfies somewhere extraction ([Definition 4.1](#)) and that Π_{FB} satisfies somewhere extraction ([Definition 4.3](#)). Then [Construction 5.3](#) satisfies statistical soundness ([Definition 5.2](#)).*

Proof. Let $\mathcal{A}$ be an unbounded adversary for the statistical soundness game of [Definition 5.2](#) against [Construction 5.3](#). Since the adversary is unbounded, we assume without loss of generality that it is deterministic. We first write the experiment out in full, with the algorithms of [Construction 5.3](#) substituted in, and introduce along the way the notation we use below.

1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^{s_c}, 1^{s_t}$ and the description of a function $\text{Ext}: [2k-1] \rightarrow \mathbb{N}$ of size at most s_t .
2. The challenger runs $\text{SoundGen}(1^\lambda, k, 1^{s_c}, 1^{s_t}, \text{Ext})$. That is, for each level $\ell = 0, \dots, h$:
 - Sample the hash parameters and a trapdoor $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$, where g_ℓ is defined in [Figure 1](#).
 - Sample binding functional BARG public parameters and a trapdoor $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$, with the same function g_ℓ .

The challenger sends $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$. The trapdoors $\text{td}_\ell^{\text{DSEH}}$ and $\text{td}_\ell^{\text{FB}}$ are not part of the output of SoundGen ([Definition 5.2](#)), they are quantities determined by its internal randomness, and we refer to them below only as bookkeeping for the hybrids.

3. The adversary $\mathcal{A}$ outputs a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s_c , a threshold $t \in \mathbb{N}$ with $t \leq k$, and a proof Π , which the challenger parses as $\Pi = (v_{\text{root}}, \text{dig}_0, \dots, \text{dig}_h, \Pi_0, \dots, \Pi_h)$.
4. The challenger computes the following quantities. They do not affect the output of the experiment, and are used only to define the hybrids below.
 - The tags $\tau_0, \dots, \tau_h$ that Verify recomputes, that is, $\tau_0 = \perp$ and $\tau_{\ell+1} = \text{dig}_\ell$, together with the hash keys $\text{hk}_\ell = \text{DSEH.Gen}(\text{pp}_\ell^{\text{DSEH}}, \tau_\ell)$ and the functional BARG common reference strings $\text{crs}_\ell^{\text{FB}} = \text{FB.Gen}(\text{pp}_\ell^{\text{FB}}, \tau_\ell)$ that it derives from them.
 - For every level $\ell \in [0, h]$, the index $p_\ell := g_\ell(\tau_\ell)$ that the hash and the functional BARG of level ℓ are binding on.
 - For every level $\ell \in [0, h]$, the block $x_\ell^* := \text{DSEH.Extract}(\text{td}_\ell^{\text{DSEH}}, \text{dig}_\ell)$ that the hash of level ℓ extracts from the digest in the proof, and the witness $\omega_\ell^* := \text{FB.Extract}(\text{td}_\ell^{\text{FB}}, \Pi_\ell)$ that the functional BARG of level ℓ extracts from the proof of that level.

5. The experiment outputs 1 if and only if all of the following hold:

- $\text{Verify}(\text{crs}, C, t, \Pi) = 1$.
- Ext is a trapdoor for the batch language of C .
- $\text{Ext}(\text{root}) < t$.

The main idea is to walk down the tree along the nodes the levels are bound on and prove that the value the proof claims for each of them exceeds the value that Ext assigns to it. We do so through a sequence of hybrid experiments that differ only in how the output of the experiment is computed. The first four steps above are unchanged throughout.

- Hyb_0 : the experiment above.
- Hyb_{j+1} for $j \in [0, h]$: the same as Hyb_j , except that the experiment outputs 1 only if, in addition to the conditions of Hyb_j , all of the following hold at level j :
 1. **Extraction**: the extracted witness satisfies the level- j circuit on the instance it is extracted at, that is, $D^{(j)}(p_j, \omega_j^*) = 1$.
 2. **Block consistency**: the block contained in ω_j^* is exactly x_j^* , and x_j^* records the index p_j . In particular x_j^* is the block of the node node_{j,p_j} .
 3. **Over-claiming**: the value that x_j^* records for node_{j,p_j} exceeds $\text{Ext}(\text{node}_{j,p_j})$.
 4. **Descent**: if $j > 0$, then node_{j,p_j} is a child of $\text{node}_{j-1,p_{j-1}}$.

In particular, Hyb_{h+1} outputs 1 only if these four conditions hold at every level. The proof now consists of the two following claims.

Claim 5.11. *There exists a negligible function negl such that for every $j \in [0, h]$,*

$$\left| \Pr[\text{Hyb}_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{j+1}(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. Throughout, fix $j \in [0, h]$ and recall that Hyb_{j+1} outputs 1 only if Hyb_j does, so that

$$\left| \Pr[\text{Hyb}_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{j+1}(\mathcal{A}) = 1] \right| = \Pr[\text{Hyb}_j(\mathcal{A}) = 1 \wedge \text{one of the four conditions at level } j \text{ fails}].$$

We first show that, on every execution on which Hyb_j outputs 1 and the extraction condition at level j holds, the three remaining conditions at level j hold as well. The difference above is then at most the probability of the event

$$\text{Bad}_j := \left[\text{Hyb}_j(\mathcal{A}) = 1 \wedge D^{(j)}(p_j, \omega_j^*) \neq 1 \right],$$

which we bound by a reduction.

Block consistency. Assume that Hyb_j outputs 1 and that $D^{(j)}(p_j, \omega_j^*) = 1$, and let x be the block contained in ω_j^* . Writing σ for the opening contained in ω_j^* and $\text{hk}_j = \text{DSEH.Gen}(\text{pp}_j^{\text{DSEH}}, \tau_j)$ for the hash key hardwired into the level- j circuit, its opening check gives

$$\text{DSEH.Verify}(\text{hk}_j, \text{dig}_j, p_j, x, \sigma) = 1.$$

Since $\text{pp}_j^{\text{DSEH}}$ was sampled by $\text{DSEH.SetupBinding}(1^\lambda, 2^j, 1^{s_f^{(j)}}, 1^{s_d^{(j)}}, g_j)$ and $p_j = g_j(\tau_j)$, somewhere extraction of Π_{DSEH} (Definition 4.1) applies to the digest dig_j and gives $x = \text{DSEH.Extract}(\text{td}_j^{\text{DSEH}}, \text{dig}_j) = x_j^*$. The indexing check of the same circuit gives that the index recorded in x is p_j . Note that this argument is unconditional: somewhere extraction of Π_{DSEH} holds perfectly and for every digest, so no probability is lost here.

Over-claiming and descent, for $j = 0$. Here $p_0 = g_0(\tau_0) = 1$, so $\text{node}_{0,p_0} = \text{root}$ and the descent condition is vacuous. The level-0 circuit is the one of Figure 3 with $\ell = 0$, so its root check gives that the value recorded in x_0^* for the root

is v_{root} . Since Hyb_0 outputs 1, the verifier accepted, and in particular its threshold check gives $v_{\text{root}} \geq t$. Since Hyb_0 outputs 1 we also have $\text{Ext}(\text{root}) < t \leq v_{\text{root}}$, which is the over-claiming condition at level 0.

Over-claiming and descent, for $j > 0$. Since Hyb_j outputs 1, the four conditions hold at level $j - 1$. By block consistency at level $j - 1$, the block x_{j-1}^* is the block contained in ω_{j-1}^* and records the index p_{j-1} . We write it as $x_{j-1}^* = (p_{j-1}, v, v_L, v_R)$, which is a well-formed inner-node block: since $j - 1 < h$, the level- $(j - 1)$ circuit is the one of Figure 3, and its first step outputs 0 on any block that does not parse as a 4-tuple. By over-claiming at level $j - 1$, we have $v > \text{Ext}(\text{node}_{j-1, p_{j-1}})$, and by the consistency check of the level- $(j - 1)$ circuit we have $v = v_L + v_R$. Write $\text{node}_L := \text{node}_{j, 2p_{j-1}-1}$ and $\text{node}_R := \text{node}_{j, 2p_{j-1}}$ for the two children of $\text{node}_{j-1, p_{j-1}}$. Since Hyb_j outputs 1, the function Ext is a trapdoor for the batch language of C , so by its propagation property $\text{Ext}(\text{node}_{j-1, p_{j-1}}) = \text{Ext}(\text{node}_L) + \text{Ext}(\text{node}_R)$. Combining these three facts, $v_L + v_R > \text{Ext}(\text{node}_L) + \text{Ext}(\text{node}_R)$, and therefore

$$v_L > \text{Ext}(\text{node}_L) \quad \text{or} \quad v_R > \text{Ext}(\text{node}_R). \quad (5.1)$$

We now determine which child the level- j instances are bound on. The function g_j of Figure 1 is invoked on the tag $\tau_j = \text{dig}_{j-1}$. It extracts the block $\text{DSEH.Extract}(\text{td}_{j-1}^{\text{DSEH}}, \text{dig}_{j-1}) = x_{j-1}^*$, parses it as (p_{j-1}, v, v_L, v_R) , and outputs $p_j = 2p_{j-1} - \beta$ where $\beta = 0$ if $v_R > \text{Ext}(\text{node}_R)$ and $\beta = 1$ otherwise. In either case $\text{node}_{j, p_j} \in \{\text{node}_L, \text{node}_R\}$ is a child of $\text{node}_{j-1, p_{j-1}}$, which is the descent condition at level j . Moreover, the value that x_{j-1}^* records for node_{j, p_j} exceeds $\text{Ext}(\text{node}_{j, p_j})$: if $\beta = 0$ then $\text{node}_{j, p_j} = \text{node}_R$ and this is the condition defining β , and if $\beta = 1$ then $\text{node}_{j, p_j} = \text{node}_L$ and it follows from Equation (5.1), whose second disjunct is ruled out by $\beta = 1$.

It remains to transfer this from the value that the parent records to the value that x_j^* records. The opening checks of the level- $(j - 1)$ circuit give an opening of the child block of node_{j, p_j} at index p_j with respect to dig_j , and its intersection check gives that the value this block records for itself is the value that x_{j-1}^* records for node_{j, p_j} . By somewhere extraction of Π_{DSEH} at level j , which is binding on the index $p_j = g_j(\tau_j)$, that block is x_j^* . Hence the value that x_j^* records for node_{j, p_j} exceeds $\text{Ext}(\text{node}_{j, p_j})$, which is the over-claiming condition at level j .

The reduction. It remains to bound $\Pr[\text{Bad}_j]$. Consider the following unbounded adversary $\mathcal{B}$ for the somewhere extraction game of Π_{FB} (Definition 4.3), which takes the level j as an input.

1. Run $\mathcal{A}(1^\lambda)$ to obtain $k, 1^{s_c}, 1^{s_t}$ and the description of Ext , and set $h = \log k$.
2. For each level $\ell = 0, \dots, h$, sample $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, 2^\ell, 1^{s_f^{(\ell)}}, 1^{s_d^{(\ell)}}, g_\ell)$, where g_ℓ is defined in Figure 1 and hardwires $\text{td}_{\ell-1}^{\text{DSEH}}$, which is available since the levels are processed from the root downwards.
3. Send $2^j, 1^{s_D}, 1^{s_f^{(j)}}$ and the description of g_j to the challenger of the somewhere extraction game, and receive public parameters, which we denote by pp_j^{FB} .
4. For each level $\ell \in [0, h] \setminus \{j\}$, sample $(\text{pp}_\ell^{\text{FB}}, \text{td}_\ell^{\text{FB}}) \leftarrow \text{FB.SetupBinding}(1^\lambda, 2^\ell, 1^{s_D}, 1^{s_f^{(\ell)}}, g_\ell)$.
5. Send $\text{crs} = (\text{pp}_0^{\text{DSEH}}, \dots, \text{pp}_h^{\text{DSEH}}, \text{pp}_0^{\text{FB}}, \dots, \text{pp}_h^{\text{FB}})$ to $\mathcal{A}$ and obtain C, t and $\Pi = (v_{\text{root}}, \text{dig}_0, \dots, \text{dig}_h, \Pi_0, \dots, \Pi_h)$.
6. Compute the tags $\tau_0, \dots, \tau_h$, the hash keys $\text{hk}_0, \dots, \text{hk}_h$ and the indices $p_0, \dots, p_h$ as in the experiment, and output the tag τ_j , the circuit $D^{(j)}[v_{\text{root}}, \text{hk}_j, \text{hk}_{j+1}, \text{dig}_j, \text{dig}_{j+1}]$ of Figure 3 if $j < h$ and the circuit $D^{(h)}[C, \text{hk}_h, \text{dig}_h]$ of Figure 2 if $j = h$, and the proof Π_j .

The simulation is perfect. Indeed, the challenger of the somewhere extraction game samples pp_j^{FB} as $\text{FB.SetupBinding}(1^\lambda, 2^j, 1^{s_D}, 1^{s_f^{(j)}}, g_j)$, which is exactly how the level- j public parameters are sampled in Hyb_j , and $\mathcal{B}$ samples every other component exactly as the challenger of Hyb_j does. The view of $\mathcal{A}$ is therefore distributed identically in the two experiments, and so are C, t, Π and the quantities τ_ℓ, p_ℓ and x_ℓ^* derived from them. Finally, the witness that the somewhere extraction challenger extracts is $\text{FB.Extract}(\text{td}_j^{\text{FB}}, \Pi_j) = \omega_j^*$, the same witness as in Hyb_j . Note that $\mathcal{B}$ never needs td_j^{FB} , which it does not hold.

The somewhere extraction experiment derives $\text{crs}_j^{\text{FB}} = \text{FB.Gen}(\text{pp}_j^{\text{FB}}, \tau_j)$ from the tag that $\mathcal{B}$ outputs, which is the same string that Verify derives for level j , and outputs 1 if and only if $\text{FB.Verify}(\text{crs}_j^{\text{FB}}, D^{(j)}, \Pi_j) = 1$ and $D^{(j)}(g_j(\tau_j), \omega_j^*) \neq 1$. This is precisely the difference in the output condition between the two hybrids: whenever Bad_j occurs,

the verifier of [Construction 5.3](#) accepted, so in particular it accepted the level- j proof, and $D^{(j)}(p_j, \omega_j^*) \neq 1$ with $p_j = g_j(\tau_j)$, hence the experiment outputs 1. We conclude that $\Pr[\text{Bad}_j] \leq \Pr[\mathcal{B}(j) \text{ wins}]$.

Finally, we obtain a single negligible function for all the levels at once. Applying the guarantee of Π_{FB} separately to each of the adversaries $\mathcal{B}(0), \dots, \mathcal{B}(h)$ would give one negligible function per level, and since the number of levels grows with λ , the maximum of these functions need not be negligible. We therefore apply it to a single adversary: let $\mathcal{B}^*$ be the adversary that samples $j \xleftarrow{\mathcal{R}} [0, h]$ and runs $\mathcal{B}(j)$. By somewhere extraction of Π_{FB} there is a negligible function negl^* with $\Pr[\mathcal{B}^* \text{ wins}] \leq \text{negl}^*(\lambda)$, and since $\mathcal{B}^*$ samples j uniformly and independently of everything else,

$$\Pr[\mathcal{B}^* \text{ wins}] = \frac{1}{h+1} \sum_{j' \in [0, h]} \Pr[\mathcal{B}(j') \text{ wins}].$$

Hence, for every level j ,

$$\Pr[\text{Bad}_j] \leq \Pr[\mathcal{B}(j) \text{ wins}] \leq \sum_{j' \in [0, h]} \Pr[\mathcal{B}(j') \text{ wins}] = (h+1) \cdot \Pr[\mathcal{B}^* \text{ wins}] \leq (h+1) \cdot \text{negl}^*(\lambda) \leq (\lambda+1) \cdot \text{negl}^*(\lambda),$$

where the last step uses $h \leq \lambda$, which holds since $k \leq 2^\lambda$. The claim follows with $\text{negl} := (\lambda+1) \cdot \text{negl}^*$, which is negligible and bounds $\Pr[\text{Bad}_j]$ for every level j . $\square$

Claim 5.12. $\Pr[\text{Hyb}_{h+1}(\mathcal{A}) = 1] = 0$.

Proof. Suppose towards a contradiction that Hyb_{h+1} outputs 1 on some execution. Then the four conditions hold at every level, and in particular at the leaves level h . By block consistency, the block x_h^* is the block contained in ω_h^* and records the index p_h . Being a leaf block, it is of the form $x_h^* = (p_h, v)$. By over-claiming, $v > \text{Ext}(\text{node}_{h, p_h})$.

By extraction, the leaf circuit of [Figure 2](#) accepts the instance p_h with the witness ω_h^* , so its range check gives $v \in \{0, 1\}$. Since Hyb_{h+1} outputs 1, the function Ext is a trapdoor for the batch language of C , so leaf soundness gives $\text{Ext}(\text{node}_{h, p_h}) \in \{0, 1\}$ as well. If $\text{Ext}(\text{node}_{h, p_h}) = 1$ then over-claiming implies $v > 1$, contradicting $v \in \{0, 1\}$. Therefore $\text{Ext}(\text{node}_{h, p_h}) = 0$, and over-claiming implies $v = 1$. The satisfiability check of the leaf circuit then gives $C(p_h, w) = 1$ for the witness w contained in ω_h^* , so the p_h^{th} instance is a yes instance. On the other hand, $\text{Ext}(\text{node}_{h, p_h}) = 0$ and leaf soundness of Ext give that the p_h^{th} instance is a no instance, that is, $C(p_h, w') = 0$ for every $w' \in \{0, 1\}^m$. This is a contradiction, so no such execution exists. $\square$

Combining [Claim 5.11](#) and [Claim 5.12](#) over the $h+1$ steps, we get

$$\Pr[\text{Hyb}_0(\mathcal{A}) = 1] \leq \Pr[\text{Hyb}_{h+1}(\mathcal{A}) = 1] + (h+1) \cdot \text{negl}(\lambda) = (h+1) \cdot \text{negl}(\lambda),$$

which is negligible since $k \leq 2^\lambda$ and therefore $h \leq \lambda$. Since Hyb_0 is the statistical soundness experiment, this completes the proof. $\square$

6 Pseudorandom Permutations with Efficient Queries

In this section, we present a mechanism for sampling pseudorandom permutations over $[N]$ with an efficient querying capability. Suppose a subset $S \subseteq [N]$ of the original elements is labeled, and let $(j_1, \dots, j_N)$ be the permutation (element i is mapped to j_i). Our goal is to support queries for the number of labeled elements in any (continuous) interval $[s, t] \subseteq [N]$:

$$\text{Query}(s, t) = |\{i \in S : j_i \in [s, t]\}|$$

Probability distributions. We denote by $\text{HS}(N, c)$ the uniform distribution over the Boolean slice $S_{N, c}$ consisting of binary strings of length N and Hamming weight c (i.e., $\text{HS}(N, c)$ corresponds to sampling a uniform binary string of length N conditioned on having Hamming weight exactly c).

Next, we recall the definition of a *hypergeometric distribution* (HGD), which describes the probability of sampling i marked elements in d draws without replacement, from a set of N total elements in which exactly c are marked. We

are interested in cases where we draw half of the elements (i.e., $d = N/2$). We denote the distribution by $\text{HGD}_{1/2}(N, c)$. Formally, the distribution $\text{HGD}_{1/2}(N, c)$ is defined via the following probability mass function

$$\Pr[\text{HGD}_{1/2}(N, c) = i] = \frac{\binom{N/2}{i} \binom{N/2}{c-i}}{\binom{N}{c}}.$$

We prove the following claim about sampling HS recursively using the hypergeometric distribution:

Claim 6.1 (Recursive Sampling of Fixed-Weight Strings). *For any even $N \in \mathbb{N}$ and $c \in [0, N]$, the distribution $\text{HS}(N, c)$ is identical to the following distribution:*

$$\mathcal{D}_{N,c} = \{\text{HS}(N/2, i) \circ \text{HS}(N/2, c-i) : i \leftarrow \text{HGD}_{1/2}(N, c)\}$$

where $\circ$ denotes concatenation.

Proof. Fix $x = x_1 \cdots x_N \in \{0, 1\}^N$ of weight c . By definition, we have

$$\Pr[\text{HS}(N, c) = x] = \binom{N}{c}^{-1}.$$

Denote $x_L = x_1 \cdots x_{N/2}$ and $x_R = x_{N/2+1} \cdots x_N$, and let i be the Hamming weight of x_L . This implies that the Hamming weight of x_R is $c - i$. By definition of $\mathcal{D}_{N,c}$, we have

$$\Pr[\mathcal{D}_{N,c} = x] = \Pr[\text{HS}(N/2, i) = x_L] \cdot \Pr[\text{HS}(N/2, c-i) = x_R] \cdot \Pr[\text{HGD}_{1/2}(N, c) = i].$$

Again, by definition of HS we have

$$\Pr[\text{HS}(N/2, i) = x_L] = \binom{N/2}{i}^{-1} \quad \text{and} \quad \Pr[\text{HS}(N/2, c-i) = x_R] = \binom{N/2}{c-i}^{-1}.$$

And by the definition of $\text{HGD}_{1/2}$, we have

$$\Pr[\text{HGD}_{1/2}(N, c) = i] = \frac{\binom{N/2}{i} \binom{N/2}{c-i}}{\binom{N}{c}}.$$

So in total we get

$$\Pr[\mathcal{D}_{N,c} = x] = \binom{N/2}{i}^{-1} \cdot \binom{N/2}{c-i}^{-1} \cdot \frac{\binom{N/2}{i} \binom{N/2}{c-i}}{\binom{N}{c}} = \binom{N}{c}^{-1}.$$

Therefore we conclude that the distribution $\mathcal{D}_{N,c}$ is identical to $\text{HS}(N, c)$, and the claim follows. $\square$

Efficient sampling of probability distributions. Goldreich, Goldwasser and Nussboim [GGN10] prove that there exists a randomized algorithm that samples from $\text{HGD}_{1/2}(N, c)$ with statistical error ε , and runs in time $\text{polylog}(N/\varepsilon)$. To match our security parameter notation, we phrase the theorem with $\lambda = -\log \varepsilon$:

Theorem 6.2 (Efficient Sampling of HGDs [GGN10, adapted from Appendix A]). *There exists an efficient algorithm Samp that takes as input a security parameter λ and an even integer $N \in \mathbb{N}$, and (not necessarily even) $c \in [0, N]$, and satisfies the two following properties:*

- **Statistical indistinguishability:** *The output distribution $\text{Samp}(1^\lambda, N, c)$ is $2^{-\lambda}$ -close to $\text{HGD}_{1/2}(N, c)$.*
- **Efficiency:** *The algorithm $\text{Samp}(1^\lambda, N, c)$ runs in $\text{poly}(\lambda, \log N)$ time.*

Throughout, we assume without loss of generality that the support of $\text{Samp}(1^\lambda, N, c)$ is a subset of the support of $\text{HGD}_{1/2}(N, c)$ for every $\lambda \in \mathbb{N}$, every even $N \in \mathbb{N}$, and every $c \in [0, N]$. Indeed, the support of $\text{HGD}_{1/2}(N, c)$ is the interval of integers $[\max(0, c - N/2), \min(N/2, c)]$, so the algorithm $\text{Samp}(1^\lambda, N, c)$ can simply map any value outside of that range to 0, without worsening the statistical distance.

Pseudorandom fixed-weight strings with efficient queries. Assuming the existence of a pseudorandom function, the algorithm from [Theorem 6.2](#) can be used to sample a binary string of length N and Hamming weight exactly c that admits an efficient procedure for computing the Hamming weight of the substring defined by the leaves of any subtree, and is computationally indistinguishable from $\text{HS}(N, c)$. Goldreich et al. [GGN10] do something similar, with the exception that the distribution they simulate is uniform over $\{0, 1\}^N$, whereas we are interested in uniform distributions over the binary strings with a fixed Hamming weight.

We capture the properties we need using the following abstraction. As in [Section 5](#), the definition is stated with respect to a tree. For simplicity, assume that N is a power of two, and let T be a complete binary tree with N leaves, whose leaves are labeled by the indices $1, \dots, N$ in left-to-right order and whose internal nodes are labeled by the indices $N + 1, \dots, 2N - 1$, using the same labeling as [Section 5](#): by increasing height and, within each height, from left to right, so that the label of each node is larger than the labels of its children. Fixing the labeling matters here, since [Query](#) below splits a count between the left and the right child, and [Construction 5.3](#) reads off the counts of specific children. We denote the label of the root by $\text{root} = 2N - 1$, and for a node $v \in [2N - 1]$ we write $\text{leaves}(v) \subseteq [N]$ for the set of leaves in the subtree of T rooted at v and $N_v := |\text{leaves}(v)|$ for their number. In particular, $\text{leaves}(j) = \{j\}$ for a leaf $j \in [N]$ and $\text{leaves}(\text{root}) = [N]$. The string $x \in \{0, 1\}^N$ is associated with the leaves of T , and a query on a node v returns the Hamming weight of x restricted to $\text{leaves}(v)$.

Definition 6.3 (Pseudorandom Fixed-Weight String). A pseudorandom fixed-weight string (PFW) generator consists of two efficient algorithms $\Pi_{\text{PFW}} = (\text{Setup}, \text{Query})$ with the following syntax:

- **Setup**($1^\lambda, N, c$) $\rightarrow K$: On input a security parameter λ , a length parameter $N \in \mathbb{N}$, and a target Hamming weight $c \in [0, N]$, both N and c given in binary, the setup algorithm outputs a key K . We assume the key contains an implicit description of N and c .
- **Query**(K, v) $\rightarrow \text{cnt} \in [0, N]$: On input a key K and a node label $v \in [2N - 1]$ (in binary), the query algorithm outputs a count $\text{cnt} \in [0, N]$. This algorithm is deterministic.
- **Correctness**: For any $\lambda \in \mathbb{N}$, any power of two $N \in \mathbb{N}$, any $c \in [0, N]$, and any K in the support of $\text{Setup}(1^\lambda, N, c)$, the following holds:
 1. **Validity**: For any leaf $j \in [N]$, we have $\text{Query}(K, j) \in \{0, 1\}$.
 2. **Consistency**: For any internal node $v \in [N + 1, 2N - 1]$ with children v_0 and v_1 , we have $\text{Query}(K, v) = \text{Query}(K, v_0) + \text{Query}(K, v_1)$.
 3. **Correct weight**: The total weight is exactly $\text{Query}(K, \text{root}) = c$.

Together, these imply that $\text{Query}(K, v) = \sum_{j \in \text{leaves}(v)} \text{Query}(K, j)$ for every node $v \in [2N - 1]$ (i.e., the query on v is the Hamming weight of the substring defined by the leaves in the subtree rooted at v).

- **Pseudorandomness**: For any adversary $\mathcal{A}$, we define the following security game parameterized by a security parameter λ and a bit $b \in \{0, 1\}$:
 1. On input a security parameter 1^λ , the adversary sends a power of two $N \in \mathbb{N}$ and a weight $c \in [0, N]$ to the challenger.
 2. The challenger samples a string $x = x_1 \cdots x_N$ as follows:
 - If $b = 0$, the challenger samples $x \leftarrow \text{HS}(N, c)$.
 - If $b = 1$, the challenger samples $K \leftarrow \text{Setup}(1^\lambda, N, c)$ and sets $x_j = \text{Query}(K, j)$ for each leaf $j \in [N]$.
 In any case, the challenger sends x to $\mathcal{A}$.
 3. The adversary outputs a bit $b' \in \{0, 1\}$ which is the output of the experiment.

We say that Π_{PFW} satisfies pseudorandomness if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

Construction 6.4 (Pseudorandom Fixed-Weight Strings). The construction simply requires the existence of a PRF $\Pi_{\text{PRF}} = (\text{PRF.Setup}, \text{PRF.Eval})$ as per Definition 3.1. By Theorem 6.2, there exists a polynomial $\ell := \text{poly}(\lambda, \log N)$ such that for any $\lambda, N \in \mathbb{N}$, the polynomial ℓ is an upper bound on the number of random bits used by $\text{Samp}(1^\lambda, N', c)$ for any $N' \leq N$ and $c \in [0, N']$. We use the tree T and the labeling described above. Intuitively, the construction assigns to every node v of T a count cnt_v , which is the Hamming weight of the substring indexed by $\text{leaves}(v)$. The root is assigned the total weight $\text{cnt}_{\text{root}} = c$, and the count of an internal node is split between its two children according to a hypergeometric sample, whose randomness is derived from the PRF evaluated on the node label. By Claim 6.1, splitting according to $\text{HGD}_{1/2}$ is exactly the way a uniform fixed-weight string distributes its weight between its two halves. Since the counts along a root-to-node path are determined by the PRF alone, the count of any node can be computed by evaluating the PRF at most $\log N$ times.

- **Setup**($1^\lambda, N, c$): On input the security parameter $\lambda \in \mathbb{N}$, a length parameter $N \in \mathbb{N}$, and a target Hamming weight $c \in [0, N]$, the setup algorithm samples a PRF key $K \leftarrow \text{PRF.Setup}(1^\lambda, 2N - 1, 1^\ell)$ and outputs (K, N, c) . Here $2N - 1$ is the domain size of the PRF (Definition 3.1), which is the number of nodes in the tree, and ℓ is its output length.
- **Query**($(K, N, c), v$): On input a key (K, N, c) consisting of a PRF key K , a length parameter $N \in \mathbb{N}$, and a target Hamming weight $c \in [0, N]$, and a node label $v \in [2N - 1]$, let $u_0, u_1, \dots, u_d$ be the path from the root to v in T (i.e., $u_0 = \text{root}$ and $u_d = v$). The query algorithm sets $\text{cnt}_{u_0} = c$ and, for each $i \in [d]$, computes the count of u_i from the count of its parent u_{i-1} as follows:
 1. Compute $r = \text{PRF.Eval}(K, u_{i-1})$ and let $z = \text{Samp}(1^\lambda, N_{u_{i-1}}, \text{cnt}_{u_{i-1}}; r)$ be the hypergeometric sample obtained using r as the randomness.
 2. If u_i is the left child of u_{i-1} , set $\text{cnt}_{u_i} = z$; otherwise set $\text{cnt}_{u_i} = \text{cnt}_{u_{i-1}} - z$.

Finally, the algorithm outputs cnt_v .

Both of our theorems refer to the counts assigned to the nodes of T , so we first fix them once and for all. Given $c \in [0, N]$ and an arbitrary split z_v for every internal node v of T , the associated *counts* $\{\text{cnt}_v\}_{v \in [2N-1]}$ are defined by the top-down pass that sets $\text{cnt}_{\text{root}} = c$ and, for every internal node v with children v_0 (left) and v_1 (right), sets $\text{cnt}_{v_0} = z_v$ and $\text{cnt}_{v_1} = \text{cnt}_v - z_v$. The following claim states that as long as every split is a legal hypergeometric value, the counts never leave their intended range.

Claim 6.5 (Counts Stay in Range). *Let $N \in \mathbb{N}$ be a power of two, let $c \in [0, N]$, and $v \in [N]$ be a leaf of the perfect binary tree T with N leaves. Fix any K in the support of $\text{PRF.Setup}(1^\lambda, N, 1^\ell)$. Let $u_0, u_1, \dots, u_d$ be the path from the root to v in T (i.e., $u_0 = \text{root}$ and $u_d = v$), and let $\{\text{cnt}_{u_i}\}_{i \in [0, d]}$ be the values computed by algorithm $\text{Query}((K, N, c), v)$. Then $\text{cnt}_{u_i} \in [0, N_{u_i}]$ for every $i \in [0, d]$, where N_{u_i} is the number of leaves in the subtree of u_i . In particular, $\text{cnt}_v \in \{0, 1\}$.*

Proof. We proceed by induction on the depth of u_i . At the root we have $\text{cnt}_{\text{root}} = c \in [0, N] = [0, N_{\text{root}}]$. Let u_i be an internal node with $\text{cnt}_{u_i} \in [0, N_{u_i}]$, and note that its two children v_0 and v_1 each have $N_{v_0} = N_{v_1} = N_{u_i}/2$ leaves, since T is perfect and N is a power of two. By definition, $\text{HGD}_{1/2}(N_{u_i}, \text{cnt}_{u_i})$ is supported on the integers j with $\binom{N_{u_i}/2}{j} \binom{N_{u_i}/2}{\text{cnt}_{u_i}-j} \neq 0$, that is, on those satisfying $0 \leq j \leq N_{u_i}/2$ and $0 \leq \text{cnt}_{u_i} - j \leq N_{u_i}/2$. Hence $\text{cnt}_{v_0} \in [0, N_{v_0}/2]$ and $\text{cnt}_{v_1} = \text{cnt}_{u_i} - \text{cnt}_{v_0} \in [0, N_{v_1}/2]$, as required. The last part follows since $N_v = 1$ for every leaf v . $\square$

Theorem 6.6 (Correctness). *Construction 6.4 satisfies correctness (Definition 6.3).*

Proof. Fix $\lambda \in \mathbb{N}$, a power of two $N \in \mathbb{N}$, a weight $c \in [0, N]$, and a key (K, N, c) in the support of $\text{Setup}(1^\lambda, N, c)$. Since PRF.Eval is deterministic and Samp is invoked on the fixed randomness $\text{PRF.Eval}(K, v)$, the split

$$z_v := \text{Samp}(1^\lambda, N_v, \text{cnt}_v; \text{PRF.Eval}(K, v))$$

at an internal node v is determined by K and cnt_v alone. Let $\{\text{cnt}_v\}_{v \in [2N-1]}$ be the counts associated with these splits. We claim that $\text{Query}((K, N, c), v) = \text{cnt}_v$ for every node $v \in [2N - 1]$. Indeed, let $\text{root} = u_0, u_1, \dots, u_d = v$ be the path from the root to v . The query algorithm sets $\text{cnt}_{u_0} = c = \text{cnt}_{\text{root}}$ and then, for each $i \in [d]$, computes the count of u_i from that of u_{i-1} using exactly the recurrence above; so by induction on i the value it assigns to u_i agrees with cnt_{u_i} , and in particular its output agrees with cnt_v . It remains to verify the three properties:

1. **Validity:** By our convention on Samp , every split satisfies $z_v \in \text{supp}(\text{HGD}_{1/2}(N_v, \text{cnt}_v))$, so [Claim 6.5](#) gives $\text{Query}((K, N, c), j) = \text{cnt}_j \in \{0, 1\}$ for every leaf $j \in [N]$.

2. **Consistency:** For an internal node v with children v_0 and v_1 , we have

$$\text{Query}((K, N, c), v_0) + \text{Query}((K, N, c), v_1) = \text{cnt}_{v_0} + \text{cnt}_{v_1} = z_v + (\text{cnt}_v - z_v) = \text{cnt}_v = \text{Query}((K, N, c), v).$$

3. **Correct weight:** The path from the root to itself has length $d = 0$, so $\text{Query}((K, N, c), \text{root}) = \text{cnt}_{\text{root}} = c$.

Since this holds for every key in the support of $\text{Setup}(1^\lambda, N, c)$, the construction satisfies correctness. $\square$

Theorem 6.7 (Pseudorandomness). *Assume that Π_{PRF} is a secure PRF ([Definition 3.1](#)) and that Samp satisfies [Theorem 6.2](#). Then [Construction 6.4](#) satisfies pseudorandomness ([Definition 6.3](#)).*

Proof. Let $\mathcal{A}$ be an efficient adversary for the pseudorandomness game and let $N \in \mathbb{N}$ and $c \in [0, N]$ be its first message, where N is a power of two. Since $\mathcal{A}$ is efficient and receives the entire string $x \in \{0, 1\}^N$, we have $N = \text{poly}(\lambda)$.

As shown in the proof of [Theorem 6.6](#), the query on a node v returns the count cnt_v associated with the splits $z_v = \text{Samp}(1^\lambda, N_v, \text{cnt}_v; \text{PRF.Eval}(K, v))$, so the counts of all nodes of T are obtained by a single top-down pass rather than by N separate root-to-leaf walks. In particular, the string sent by the challenger in the case $b = 1$ is $x = x_1 \cdots x_N$ with $x_j = \text{cnt}_j$ for every leaf $j \in [N]$. Note that only the $N - 1$ internal nodes are ever fed to the PRF. We define the following sequence of hybrid experiments; each is described as a modification of the previous one, and all of them output $\mathcal{A}$'s guess b' .

- Hyb_0 : the pseudorandomness game with $b = 1$. In full:
 1. $\mathcal{A}(1^\lambda)$ outputs a power of two $N \in \mathbb{N}$ and a weight $c \in [0, N]$.
 2. The challenger samples $K \leftarrow \text{PRF.Setup}(1^\lambda, 2N - 1, 1^\ell)$, sets $\text{cnt}_{\text{root}} = c$, and computes the counts of all nodes in a top-down pass, where the split at an internal node v is $z_v = \text{Samp}(1^\lambda, N_v, \text{cnt}_v; \text{PRF.Eval}(K, v))$. It sends $x = x_1 \cdots x_N$ with $x_j = \text{cnt}_j$ to $\mathcal{A}$.
 3. $\mathcal{A}$ outputs b' , the output of the experiment.
- Hyb_1 : same as Hyb_0 , except the challenger replaces the PRF by a truly random function: it samples a uniform $R: \{0, 1\}^{1+\log N} \rightarrow \{0, 1\}^\ell$ and sets $z_v = \text{Samp}(1^\lambda, N_v, \text{cnt}_v; R(v))$ for every internal node v .
- Hyb_2 : same as Hyb_1 , except every split is an exact hypergeometric sample: for each internal node v , the challenger samples $z_v \leftarrow \text{HGD}_{1/2}(N_v, \text{cnt}_v)$, independently across nodes.
- Hyb_3 : the pseudorandomness game with $b = 0$; that is, the challenger samples $x \leftarrow \text{HS}(N, c)$ and sends it to $\mathcal{A}$.

We now argue that consecutive hybrids are indistinguishable.

Claim 6.8. *Assume Π_{PRF} is a secure PRF. Then for any efficient $\mathcal{A}$ there exists a negligible function negl such that $|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_1(\mathcal{A}) = 1]| \leq \text{negl}(\lambda)$.*

Proof. The experiments differ only in whether the randomness of the splits is $\text{PRF.Eval}(K, v)$ or $R(v)$. We build an adversary $\mathcal{B}$ for the PRF pseudorandomness game. On input 1^λ , the adversary $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain N and c , and sends the input length $1^{1+\log N}$ and the output length 1^ℓ to the PRF challenger. It then performs the top-down pass: it sets $\text{cnt}_{\text{root}} = c$ and, for each internal node $v \in [N+1, 2N-1]$ in decreasing order of label, it queries its oracle on v to receive r_v and splits cnt_v using $z_v = \text{Samp}(1^\lambda, N_v, \text{cnt}_v; r_v)$. Finally, $\mathcal{B}$ sends $x = x_1 \cdots x_N$ with $x_j = \text{cnt}_j$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Each internal node is queried exactly once, so all $N - 1 = \text{poly}(\lambda)$ queries are distinct and $\mathcal{B}$ is efficient. If the oracle is $\text{PRF.Eval}(K, \cdot)$, then $\mathcal{B}$ perfectly simulates Hyb_0 , and if it returns uniform strings on distinct inputs, then $\mathcal{B}$ perfectly simulates Hyb_1 . Thus the advantage of $\mathcal{B}$ is exactly the claimed difference, which is negligible. $\square$

Claim 6.9. *For any (potentially unbounded) $\mathcal{A}$, we have $|\Pr[\text{Hyb}_1(\mathcal{A}) = 1] - \Pr[\text{Hyb}_2(\mathcal{A}) = 1]| \leq N \cdot 2^{-\lambda}$.*

Proof. Since R is a uniform random function and each internal node is evaluated once, the strings $\{R(v)\}_{v \in [N+1, 2N-1]}$ are independent and uniform over $\{0, 1\}^\ell$. Hence, conditioned on cnt_v , the split z_v in Hyb_1 is distributed as $\text{Samp}(1^\lambda, N_v, \text{cnt}_v)$, independently across internal nodes, whereas in Hyb_2 it is distributed as $\text{HGD}_{1/2}(N_v, \text{cnt}_v)$. By [Theorem 6.2](#), these two distributions are $2^{-\lambda}$ -close for every v , so replacing the splits one internal node at a time and applying the triangle inequality over the $N - 1$ internal nodes, the statistical distance between the strings x produced in Hyb_1 and in Hyb_2 is at most $(N - 1) \cdot 2^{-\lambda} \leq N \cdot 2^{-\lambda}$. As x is the only message $\mathcal{A}$ receives, the claim follows. $\square$

Claim 6.10. *For any (potentially unbounded) $\mathcal{A}$, we have $\Pr[\text{Hyb}_2(\mathcal{A}) = 1] = \Pr[\text{Hyb}_3(\mathcal{A}) = 1]$.*

Proof. It suffices to show that the string x sampled in Hyb_2 is distributed exactly as $\text{HS}(N, c)$. We show the following more general statement by induction on the height of the subtree: for every node v of T and every $w \in [0, N_v]$, conditioned on $\text{cnt}_v = w$, the substring $x|_{\text{leaves}(v)}$ is distributed according to $\text{HS}(N_v, w)$. Applying this to the root, where $\text{cnt}_{\text{root}} = c$ and $\text{leaves}(\text{root}) = [N]$, gives the claim.

For the base case, let v be a leaf, so $N_v = 1$ and, by [Claim 6.5](#), $\text{cnt}_v = w \in \{0, 1\}$. By construction $x|_{\text{leaves}(v)} = x_v = w$, which is exactly the distribution $\text{HS}(1, w)$.

For the inductive step, let v be an internal node with children v_0 and v_1 , and condition on $\text{cnt}_v = w$. The challenger samples $z_v \leftarrow \text{HGD}_{1/2}(N_v, w)$ and sets $\text{cnt}_{v_0} = z_v$ and $\text{cnt}_{v_1} = w - z_v$. The splits inside the subtrees rooted at v_0 and at v_1 use independent samples associated with disjoint sets of nodes, so conditioned on z_v , the substrings $x|_{\text{leaves}(v_0)}$ and $x|_{\text{leaves}(v_1)}$ are independent, and by the induction hypothesis they are distributed according to $\text{HS}(N_v/2, z_v)$ and $\text{HS}(N_v/2, w - z_v)$, respectively. Since the leaves of T are labeled left to right, we have $x|_{\text{leaves}(v)} = x|_{\text{leaves}(v_0)} \circ x|_{\text{leaves}(v_1)}$, and therefore

$$x|_{\text{leaves}(v)} \sim \left\{ \text{HS}(N_v/2, i) \circ \text{HS}(N_v/2, w - i) : i \leftarrow \text{HGD}_{1/2}(N_v, w) \right\} = \mathcal{D}_{N_v, w}.$$

By [Claim 6.1](#), applied with the parameters N_v and w , the distribution $\mathcal{D}_{N_v, w}$ is identical to $\text{HS}(N_v, w)$, which completes the induction. $\square$

Combining [Claim 6.8](#), [Claim 6.9](#), and [Claim 6.10](#), and recalling that $N = \text{poly}(\lambda)$, we conclude that for every efficient adversary $\mathcal{A}$,

$$|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1]| \leq \text{negl}(\lambda) + N \cdot 2^{-\lambda} = \text{negl}(\lambda).$$

Since Hyb_0 and Hyb_3 are the pseudorandomness game with $b = 1$ and with $b = 0$, respectively, [Construction 6.4](#) satisfies pseudorandomness. $\square$

Pseudorandom permutations. As mentioned earlier, we are interested in sampling pseudorandom permutations over $[N]$. For example, the classic Fisher-Yates shuffle [[FY53](#)] produces a random permutation given access to a uniform distribution over $[N]$, which in turn can be simulated with $\text{negl}(\lambda)$ error using $\text{poly}(\lambda, \log N)$ uniform random bits.

We describe an alternative procedure to sample a permutation over $[N]$. Intuitively, the procedure takes as input N and a subset $S \subseteq [N]$ of size c and proceeds in two phases:

1. Randomly samples a subset $\mathcal{I} \subseteq [N]$ of size c , which is equivalent to sampling a random fixed-weight string.
2. Randomly assigns the elements of S to indices in $\mathcal{I}$, and randomly assigns elements of the complement $\bar{S}$ to indices in $\bar{\mathcal{I}}$.

Formally, the procedure is described in the following figure:

Inputs: A set size parameter 1^N , and a subset $S \subseteq [N]$ of size c .

1. Sample $x_1 \cdots x_N \leftarrow \text{HS}(N, c)$ (i.e., a uniform binary string of length N with Hamming weight exactly c).
2. Let $\{j_1^{(0)}, \dots, j_{N-c}^{(0)}\}$ be the subset of indices corresponding to the zeroes of x , and $\{j_1^{(1)}, \dots, j_c^{(1)}\}$ be the subset of indices corresponding to the ones of x , in some fixed order.
3. Sample a random bijection $\tau^{(0)} : \bar{S} \rightarrow [N - c]$ and a random bijection $\tau^{(1)} : S \rightarrow [c]$.
4. For each $i \in \bar{S}$, set $j_i = j_{\tau^{(0)}(i)}^{(0)}$.
5. For each $i \in S$, set $j_i = j_{\tau^{(1)}(i)}^{(1)}$.
6. Output $(j_1, \dots, j_N)$.

Figure 4: Algorithm Permute^* .

Proposition 6.11 (Correctness of Permute^*). *Algorithm $\text{Permute}^*(1^N, S)$ outputs a random permutation.*

Proof. Fix any $N \in \mathbb{N}$ and $S \subseteq [N]$, and let $(j'_1, \dots, j'_N)$ be a permutation over $[N]$ and for each $i \in [N]$ let

$$y_{j'_i} = \begin{cases} 1 & i \in S \\ 0 & i \notin S \end{cases}$$

We want to prove the following

$$\Pr[(j'_1, \dots, j'_N) = \text{Permute}^*(1^N, S)] = 1/N!$$

Consider a run of $\text{Permute}^*(1^N, S)$, and denote by x , $\tau^{(0)}$ and $\tau^{(1)}$ the string and the two bijections it samples, and by $(j_1, \dots, j_N)$ its output. By steps 4 and 5, $x_{j_i} = 1$ if $i \in S$ and $x_{j_i} = 0$ otherwise, so $(j_1, \dots, j_N) = (j'_1, \dots, j'_N)$ implies $x = y$. Condition on $x = y$. For $i \in S$, the index j'_i is a one of x , so $j'_i = j_n^{(1)}$ for a unique $n \in [c]$, and $j_i = j'_i$ if and only if $\tau^{(1)}(i) = n$. Ranging over $i \in S$ determines $\tau^{(1)}$, and the same argument over $i \in \bar{S}$ and the zeroes of x determines $\tau^{(0)}$. The string and the two bijections are sampled independently, x is uniform over the $\binom{N}{c}$ strings of length N and Hamming weight c , and y is one of them, so

$$\Pr[(j'_1, \dots, j'_N) = \text{Permute}^*(1^N, S)] = \frac{1}{\binom{N}{c}} \cdot \frac{1}{c!} \cdot \frac{1}{(N-c)!} = \frac{1}{N!}. \quad \square$$

7 Puncturable Public-Coin Threshold Signatures

In this section, we define and construct our notion of puncturable threshold signatures. Recall that a threshold signature system has k individual signing keys $sk_1, \dots, sk_k$, a single (large) aggregation key ak , and a single succinct verification key vk_{agg} . The aggregation key ak allows combining multiple signatures into a single, succinct signature σ_{agg} , and the verification key vk_{agg} can be used to efficiently verify that σ_{agg} was produced by aggregating at least t valid signatures. We require that verifying the aggregated signature does not scale linearly with the number of aggregated signatures. We say that the scheme is puncturable if for any message m^* and a subset $S \subseteq [k]$, it is possible to produce an aggregate verification key that does not accept any signature over the threshold $|S|$ on the message m^* . Note that this is a statistical property (that holds with overwhelming probability of the coins used to produce the aggregate key).

Here we consider a distributed variant where users can generate their own individual signing key and verification key pairs. The simulator generates keys only for uncorrupted users; the adversary supplies arbitrary verification keys for corrupted users. Then there exists a public-coin randomized algorithm that aggregates a collection of verification keys to produce the aggregation key and master verification key for the quorum. As we discuss in [Remark 7.2](#), this

syntax allows us to obtain a centralized scheme in the plain model (where the trusted authority runs the aggregation algorithm) as well as a scheme with silent setup in the random oracle model (where the aggregation randomness is derived from the random oracle).

Definition 7.1 (Puncturable Public-Coin Threshold Signatures). A puncturable public-coin threshold signature scheme Π_{PPCTS} for a message space $\mathcal{M}$ is a tuple of efficient algorithms

$$\Pi_{\text{PPCTS}} = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{PartialVerify}, \text{Preprocess}, \text{Aggregate}, \text{Verify}, \text{Sim})$$

with the following syntax:

- $\text{Setup}(1^\lambda) \rightarrow \text{crs}$: On input a security parameter λ , the setup algorithm outputs a common reference string crs .
- $\text{KeyGen}(1^\lambda, \text{crs}) \rightarrow (\text{vk}, \text{sk})$: On input a security parameter λ and a common reference string crs , the key-generation algorithm outputs a verification key vk and a signing key sk .
- $\text{Sign}(\text{sk}, m) \rightarrow \sigma$: On input a signing key sk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ .
- $\text{PartialVerify}(\text{vk}, m, \sigma) \rightarrow b$: On input a verification key vk , a message $m \in \mathcal{M}$, and a signature σ , the partial-verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.
- $\text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k)) \rightarrow (\text{vk}_{\text{agg}}, \text{ak})$: On input a common reference string crs and a vector of verification keys $(\text{vk}_1, \dots, \text{vk}_k)$ of size $k \in [2^\lambda]$, the preprocessing algorithm outputs an aggregated verification key vk_{agg} and an aggregation key ak . This algorithm is randomized.
- $\text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T}) \rightarrow \sigma_{\text{agg}}$: On input the aggregation key ak , a message $m \in \mathcal{M}$, and a set of index and signature pairs $\{(\ell, \sigma_\ell)\}_{\ell \in T}$ for $T \subseteq [k]$, the aggregation algorithm outputs an aggregate signature σ_{agg} . This algorithm is deterministic.
- $\text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}}) \rightarrow b$: Given an aggregated verification key vk_{agg} , a threshold t , a message $m \in \mathcal{M}$, and an aggregate signature σ_{agg} , verification outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.
- $\text{Sim}(1^\lambda, 1^k, S, m^*) \rightarrow (\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r)$: On input a security parameter λ , a number of parties $k \in [2^\lambda]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$, the simulator outputs a common reference string crs , a key pair $(\text{vk}_\ell, \text{sk}_\ell)$ for each honest party $\ell \in [k] \setminus S$, and randomness r for Preprocess. The challenger retains the honest signing keys to answer signing queries; the simulator generates no keys for corrupted parties.

We require that Π_{PPCTS} satisfy the following properties:

- **Signing correctness:** For all $\lambda \in \mathbb{N}$, all crs in the support of $\text{Setup}(1^\lambda)$, and all messages $m \in \mathcal{M}$,

$$\Pr \left[\text{PartialVerify}(\text{vk}, m, \sigma) = 1 : \begin{array}{l} (\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda, \text{crs}) \\ \sigma \leftarrow \text{Sign}(\text{sk}, m) \end{array} \right] = 1.$$

- **Aggregation correctness:** For all $\lambda \in \mathbb{N}$, all $k \in [2^\lambda]$, all messages $m \in \mathcal{M}$, all thresholds $t \in [k]$, all sets $T \subseteq [k]$ with $|T| \geq t$, all crs in the support of $\text{Setup}(1^\lambda)$, all $\{(\text{vk}_\ell, \text{sk}_\ell)\}_{\ell \in [k]}$ where $(\text{vk}_\ell, \text{sk}_\ell)$ is in the support of $\text{KeyGen}(1^\lambda, \text{crs})$ for all $\ell \in [k]$, and all signatures $\{\sigma_\ell\}_{\ell \in T}$ where, for every $\ell \in T$, $\text{PartialVerify}(\text{vk}_\ell, m, \sigma_\ell) = 1$,

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}}) = 1 : \\ (\text{vk}_{\text{agg}}, \text{ak}) \leftarrow \text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k)) \\ \sigma_{\text{agg}} \leftarrow \text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T}) \end{array} \right] = 1.$$

- **Punctured static security:** For a security parameter λ and a (potentially unbounded) adversary $\mathcal{A}$, we define the punctured static security game as follows:

1. On input the security parameter 1^λ , algorithm $\mathcal{A}$ outputs a number of parties 1^k where $k \in [2^\lambda]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$.
2. The challenger samples $(\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r) \leftarrow \text{Sim}(1^\lambda, 1^k, S, m^*)$ and gives crs together with $(\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$. Revealing r makes the game public-coin.
3. Throughout the game, algorithm $\mathcal{A}$ may adaptively make signing queries: on a query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, the challenger replies with $\text{Sign}(\text{sk}_\ell, m)$.
4. Algorithm $\mathcal{A}$ outputs arbitrary verification keys $(\text{vk}_\ell)_{\ell \in S}$ for the corrupted users (not necessarily generated by KeyGen), a threshold $t \in [k]$ with $t > |S|$, and a forgery σ_{agg} .
5. The challenger combines the simulated honest verification keys with the adversary's corrupted verification keys and computes $(\text{vk}_{\text{agg}}, \text{ak}) \leftarrow \text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k); r)$. The output of the experiment is $b = \text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}})$.

We say Π_{PPCTS} satisfies punctured static security if for all (potentially unbounded) adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] \leq \text{negl}(\lambda)$ in the punctured static security game defined above.

- **Public-coin mode indistinguishability:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the mode indistinguishability game as follows:

1. On input the security parameter 1^λ , algorithm $\mathcal{A}$ outputs a number of parties 1^k where $k \in [2^\lambda]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$.
2. The challenger computes crs , for each honest party $\ell \in [k] \setminus S$, a key pair $(\text{vk}_\ell, \text{sk}_\ell)$, and randomness r for Preprocess as follows:
 - If $b = 0$, it samples $\text{crs} \leftarrow \text{Setup}(1^\lambda)$, for each $\ell \in [k] \setminus S$, $(\text{vk}_\ell, \text{sk}_\ell) \leftarrow \text{KeyGen}(1^\lambda, \text{crs})$, and r uniformly at random.
 - If $b = 1$, it samples $(\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r) \leftarrow \text{Sim}(1^\lambda, 1^k, S, m^*)$.
It gives crs together with $(\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$.
3. Throughout the game, algorithm $\mathcal{A}$ may adaptively make signing queries: on a query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, the challenger replies with $\text{Sign}(\text{sk}_\ell, m)$.
4. Algorithm $\mathcal{A}$ outputs a guess b' , which is the output of the experiment.

We say that Π_{PPCTS} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Succinctness:** There exist universal polynomials $p_{\text{vk}}(\cdot, \cdot)$ and $p_{\text{agg}}(\cdot, \cdot)$ such that for all $\lambda \in \mathbb{N}$, all $k \in [2^\lambda]$, all crs in the support of $\text{Setup}(1^\lambda)$, all $t \in [k]$, all $(\text{vk}_\ell, \text{sk}_\ell)$ in the support of $\text{KeyGen}(1^\lambda, \text{crs})$ for $\ell \in [k]$, and all $(\text{vk}_{\text{agg}}, \text{ak})$ in the support of $\text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k))$, the following hold:
 - The size of the aggregated verification key vk_{agg} is at most $p_{\text{vk}}(\lambda, \log k)$.
 - The size of the aggregate signatures output by $\text{Aggregate}(\text{ak}, \cdot, \cdot)$ is at most $p_{\text{agg}}(\lambda, \log k)$.

Verification via $\text{Verify}(\text{vk}_{\text{agg}}, \cdot, \cdot, \cdot)$ therefore takes $\text{poly}(\lambda, \log k)$ time.

Remark 7.2 (Centralized and Silent Variants). Note that a puncturable public-coin threshold signature scheme immediately implies a centralized threshold signature scheme in the standard model. Namely, the central authority can sample k key-pairs itself, preprocess them to generate a short master verification key and a long aggregation key. More interestingly, a public-coin threshold encryption scheme yields a silent threshold signature scheme in the random oracle model. The only difference between the two primitives is that the latter requires the preprocessing algorithm to be deterministic so that the same verification and aggregation keys can be independently computed by any party without additional communication. The random oracle can be used to derive the randomness for the preprocessing algorithm, and thus “derandomize” it. We sketch the security reduction in [Section 10](#).

7.1 Our Construction

We now give our construction of a puncturable public-coin threshold signature scheme. We refer to [Section 2.1](#) for an overview of this construction.

Construction 7.3 (Puncturable Public-Coin Threshold Signatures). Our construction of puncturable public-coin threshold signatures uses the following cryptographic ingredients:

- An all-but-one signature scheme $\Pi_{\text{ABO}} = (\text{ABO.Setup}, \text{ABO.SetupPunc}, \text{ABO.Sign}, \text{ABO.Verify})$ as per [Definition 3.3](#), with message space $\mathcal{M}$. Let $\ell_{\text{ABO}} := \ell_{\text{ABO}}(\lambda)$ be the length of a signature in its signature space, let $\ell_{\text{vk}_{\text{ABO}}} := \ell_{\text{vk}_{\text{ABO}}}(\lambda)$ be the length of a verification key in the support of $\text{ABO.Setup}(1^\lambda)$, and let $\ell_{r'} := \ell_{r'}(\lambda)$ be the length of the randomness consumed by ABO.SetupPunc . We assume that there exists a special value $\text{vk}_\perp$ that is punctured on all messages (i.e., for all m, σ it holds $\text{ABO.Verify}(\text{vk}_\perp, m, \sigma) = 0$).
- A function-binding hash for conjunctions $\Pi_{\text{FBH}} = (\text{FBH.Setup}, \text{FBH.SetupBinding}, \text{FBH.Hash}, \text{FBH.Verify})$ as per [Definition 3.5](#), over the block space $\mathcal{X} = [2^\lambda] \times \{0, 1\}^{\ell_{\text{vk}_{\text{ABO}}}}$, whose blocks pair a slot with an all-but-one verification key; taking the index range to be $[2^\lambda]$ rather than the number of blocks keeps $\mathcal{X}$ independent of it, as [Definition 3.5](#) requires. Let ℓ_ρ be a bound on the length of an opening produced by FBH.Hash , and let $s_g^{(n)}$ be a bound, for batch size 2^n , on the description size of the predicate that FBH.SetupBinding will bind on to argue mode indistinguishability; we pin down its value only once we fix that mechanism.
- A BARG for trapdoor threshold languages $\Pi_{\text{BARG}} = (\text{BARG.Gen}, \text{BARG.SoundGen}, \text{BARG.Prove}, \text{BARG.Verify})$ as per [Definition 5.2](#).
- A pseudorandom fixed-weight string generator $\Pi_{\text{PFW}} = (\text{PFW.Setup}, \text{PFW.Query})$ as per [Definition 6.3](#). None of the algorithms above call Π_{PFW} ; it is used only when we argue punctured static security and mode indistinguishability ([Definition 7.1](#)), where a circuit computing $v \mapsto \text{PFW.Query}(K, v)$ with a key $K \leftarrow \text{PFW.Setup}(1^\lambda, 2^n, c)$ hardwired, for $n \in [\lambda]$ and $c \in [0, 2^n]$, will describe the BARG trapdoor Ext that marks the slots held by a corrupted party, leaving every other slot to be a no-instance; this is what eventually pins down $s_t^{(n)}$.
- A PRF $\Pi_{\text{PRF}} = (\text{PRF.Setup}, \text{PRF.Eval})$ as per [Definition 3.1](#), with input length $n \in [\lambda]$ and output length $\ell_{r'}$. Like Π_{PFW} , it is used only in that same analysis, to derandomize the punctured all-but-one keys sampled for the honest slots.

The construction uses the following circuit:

Constants: A batch size exponent $n \in [\lambda]$, a hash key hk , a digest dig , and a message $m \in \mathcal{M}$.
Inputs: An index $j \in [2^n]$ as the instance, and a witness $(\text{vk}, \rho, \sigma)$.

1. Output 0 unless $\text{FBH.Verify}(\text{hk}, \text{dig}, j, (j, \text{vk}), \rho) = 1$.
2. Output 1 if and only if $\text{ABO.Verify}(\text{vk}, m, \sigma) = 1$.

Figure 5: Circuit $C[n, \text{hk}, \text{dig}, m]$ for the index BARG.

The predicate below is what FBH.SetupBinding binds on when Sim places Π_{FBH} in binding mode.

Constants: A PFW key K , a PRF key K_{PRF} , an index space size N , and a message $m^* \in \mathcal{M}$.

Inputs: A block $(i, \text{vk}) \in [2^\lambda] \times \{0, 1\}^{\ell_{\text{vkABO}}}$.

1. Output 1 if $i \notin [N]$.
2. Otherwise output 1 if and only if $\text{PFW.Query}(K, i) = 1$, or $\text{vk} = \text{vk}_\perp$, or vk is the second component (the verification key) of the output of $\text{ABO.SetupPunc}(1^\lambda, m^*; \text{PRF.Eval}(K_{\text{PRF}}, i))$.

Figure 6: Circuit $g[K, K_{\text{PRF}}, m^*]$ for the function-binding hash.

The algorithms for the scheme are defined as follows:

- **Setup(1^λ):** On input a security parameter λ , the setup algorithm proceeds as follows:
 1. For each $n \in [\lambda]$:
 - (a) Let $s_g^{(n)} := s_g^{(n)}(\lambda, n)$ be a bound on the size of the circuit of Figure 6 with security parameter λ and index space size 2^n , let $s_c^{(n)} := s_c(\lambda, n)$ be a bound on the size of the circuit of Figure 5 with security parameter λ and index space size 2^n , and let $s_t^{(n)} := s_t^{(n)}(\lambda, n)$ be the size of the circuit computing PFW.Query with security parameter λ and string length 2^n .
 - (b) Sample a hash key $\text{hk}^{(n)} \leftarrow \text{FBH.Setup}(1^\lambda, 2^n, 1^{s_g^{(n)}})$.
 - (c) Sample a BARG normal setup $\text{crs}_{\text{BARG}}^{(n)} \leftarrow \text{BARG.Gen}(1^\lambda, 2^n, 1^{s_c^{(n)}}, 1^{s_t^{(n)}})$.
 2. Output $\text{crs} = (\text{hk}^{(n)}, \text{crs}_{\text{BARG}}^{(n)})_{n \in [\lambda]}$.
- **KeyGen($1^\lambda, \text{crs}$):** On input a security parameter λ and a common reference string crs , the key-generation algorithm proceeds as follows:
 1. Sample an all-but-one key pair $(\text{sk}^{\text{ABO}}, \text{vk}^{\text{ABO}}) \leftarrow \text{ABO.Setup}(1^\lambda)$.
 2. Output $\text{vk} := \text{vk}^{\text{ABO}}$ and $\text{sk} := \text{sk}^{\text{ABO}}$.
- **Sign(sk, m):** On input a signing key sk and a message $m \in \mathcal{M}$, the signing algorithm outputs $\sigma \leftarrow \text{ABO.Sign}(\text{sk}, m)$.
- **PartialVerify(vk, m, σ):** On input a verification key vk , a message $m \in \mathcal{M}$, and a signature σ , the partial-verification algorithm outputs $\text{ABO.Verify}(\text{vk}, m, \sigma)$.
- **Preprocess($\text{crs}, (\text{vk}_1, \dots, \text{vk}_k)$):** On input a common reference string $\text{crs} = (\text{hk}^{(n')}, \text{crs}_{\text{BARG}}^{(n')})_{n' \in [\lambda]}$ and a vector of verification keys $(\text{vk}_1, \dots, \text{vk}_k)$ of size $k \in [2^\lambda]$, the preprocessing algorithm proceeds as follows:
 1. If $k = 1$, output $\text{vk}_{\text{agg}} := \text{vk}_1$ and $\text{ak} := \perp$.
 2. Let $n \in [\lambda]$ be the unique exponent with $k \in [2^{n-1} + 1, 2^n]$, and denote $N := 2^n$, $\text{hk} := \text{hk}^{(n)}$ and $\text{crs}_{\text{BARG}} := \text{crs}_{\text{BARG}}^{(n)}$.
 3. For each $\ell \in [k + 1, N]$, set $\text{vk}_\ell := \text{vk}_\perp$ to be the rejecting key.
 4. Sample a uniformly random permutation $(j_1, \dots, j_N)$ of $[N]$, assigning slot j_ℓ to party ℓ . (This permutation is the only randomness Preprocess consumes, so we identify Preprocess's random coins r with the permutation itself.)
 5. Hash the verification keys in slot order,

$$(\text{dig}, \rho_1, \dots, \rho_N) = \text{FBH.Hash}(\text{hk}, (x_1, \dots, x_N)),$$

where $x_{j_\ell} := (\text{vk}_\ell, \text{vk}_\ell)$ for each $\ell \in [N]$.

6. Output $\text{vk}_{\text{agg}} := (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}})$ and $\text{ak} := (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}}, \{(\text{vk}_\ell, j_\ell, \rho_{j_\ell})\}_{\ell \in [k]})$.

- $\text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T})$: On input the aggregation key ak , which is either $\perp$ or

$$\text{ak} = (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}}, \{(\text{vk}_\ell, j_\ell, \rho_{j_\ell})\}_{\ell \in [k]}),$$

a message $m \in \mathcal{M}$, and a set of index and signature pairs $\{(\ell, \sigma_\ell)\}_{\ell \in T}$ for $T \subseteq [k]$, the aggregation algorithm proceeds as follows:

1. If $\text{ak} = \perp$, output $\sigma_{\text{agg}} := \sigma_1$ when $T = \{1\}$ and $\sigma_{\text{agg}} := \perp$ when $T = \emptyset$.
 2. Let $C := C[n, \text{hk}, \text{dig}, m]$ be the circuit of Figure 5.
 3. Let $S_{\text{BARG}} := \{j_\ell\}_{\ell \in T}$ be the set of slots assigned to the parties in T , set $w_{j_\ell} := (\text{vk}_\ell, \rho_{j_\ell}, \sigma_\ell)$ for each $\ell \in T$, and output $\sigma_{\text{agg}} \leftarrow \text{BARG.Prove}(\text{crs}_{\text{BARG}}, C, S_{\text{BARG}}, (w_j)_{j \in S_{\text{BARG}}})$.
- $\text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}})$: On input an aggregated verification key vk_{agg} , which is either a single all-but-one verification key or $\text{vk}_{\text{agg}} = (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}})$, a threshold t , a message $m \in \mathcal{M}$, and an aggregate signature σ_{agg} , the verification algorithm proceeds as follows:

1. If vk_{agg} is a single all-but-one verification key rather than a 4-tuple, output 1 if and only if $t \leq 1$ and $\text{PartialVerify}(\text{vk}_{\text{agg}}, m, \sigma_{\text{agg}}) = 1$.
2. Let $C := C[n, \text{hk}, \text{dig}, m]$ be the circuit of Figure 5.
3. Output $\text{BARG.Verify}(\text{crs}_{\text{BARG}}, C, t, \sigma_{\text{agg}})$.

- $\text{Sim}(1^\lambda, 1^k, S, m^*)$: On input a security parameter λ , a number of parties $k \in [2^\lambda]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$, the simulator proceeds as follows:

1. If $k = 1$, sample $\text{crs} \leftarrow \text{Setup}(1^\lambda)$ and, if $S = \emptyset$, set $(\text{sk}_1, \text{vk}_1) \leftarrow \text{ABO.SetupPunc}(1^\lambda, m^*)$. Output $\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [1] \setminus S}$ (the empty tuple if $S = \{1\}$), and the empty randomness $r := \perp$, which is all Preprocess consumes on a single key.
2. Let $n \in [\lambda]$ be the unique exponent with $k \in [2^{n-1} + 1, 2^n]$, and denote $N := 2^n$, $\bar{S} := [N] \setminus S$ and $c := |S|$. The slots of $\bar{S}$ are the ones Preprocess will fill with a key that no signature on m^* verifies against: a key punctured at m^* for an honest party $\ell \in \bar{S} \cap [k]$, and $\text{vk}_\perp$ for a padding slot $\ell \in [k + 1, N]$.
3. Sample $K \leftarrow \text{PFW.Setup}(1^\lambda, N, c)$ and $K_{\text{PRF}} \leftarrow \text{PRF.Setup}(1^\lambda, N, 1^{\ell'})$.
4. Sample $(j_1, \dots, j_N) \leftarrow \text{Permute}^*(1^N, S)$ (Figure 4), except that the string x inside Permute^* is set to $x_j := \text{PFW.Query}(K, j)$ for each $j \in [N]$, instead of being sampled from $\text{HS}(N, c)$. Writing $\mathcal{I} := \{j \in [N] : \text{PFW.Query}(K, j) = 1\}$ for the resulting marked set, this gives $\{j_\ell\}_{\ell \in S} = \mathcal{I}$, so the marked slots are exactly those held by the corrupted parties.
5. For each $\ell \in \bar{S} \cap [k]$, set $(\text{sk}_\ell, \text{vk}_\ell) := \text{ABO.SetupPunc}(1^\lambda, m^*; \text{PRF.Eval}(K_{\text{PRF}}, j_\ell))$.
6. For each $n' \in [\lambda] \setminus \{n\}$:
 - (a) Let $s_g^{(n')} := s_g^{(n')}(\lambda, n')$ be a bound on the size of the circuit of Figure 6 with security parameter λ and index space size $2^{n'}$, let $s_c^{(n')} := s_c^{(n')}(\lambda, n')$ be a bound on the size of the circuit of Figure 5 with security parameter λ and index space size $2^{n'}$, and let $s_t^{(n')} := s_t^{(n')}(\lambda, n')$ be the size of the circuit computing PFW.Query with security parameter λ and string length $2^{n'}$.
 - (b) Sample a hash key $\text{hk}^{(n')} \leftarrow \text{FBH.Setup}(1^\lambda, 2^{n'}, 1^{s_g^{(n')}})$.
 - (c) Sample a BARG normal setup $\text{crs}_{\text{BARG}}^{(n')} \leftarrow \text{BARG.Gen}(1^\lambda, 2^{n'}, 1^{s_c^{(n')}} , 1^{s_t^{(n')}})$.
7. Sample a binding hash key $\text{hk}^{(n)} \leftarrow \text{FBH.SetupBinding}(1^\lambda, N, 1^{s_g^{(n)}} , g)$, where the predicate $g := g[K, K_{\text{PRF}}, m^*]$ is defined in Figure 6.
8. Sample $\text{crs}_{\text{BARG}}^{(n)} \leftarrow \text{BARG.SoundGen}(1^\lambda, N, 1^{s_c^{(n)}} , 1^{s_t^{(n)}} , \text{Ext})$, where $\text{Ext}(v) := \text{PFW.Query}(K, v)$ for every $v \in [2N - 1]$.

9. Output $\text{crs} = (\text{hk}^{(n')}, \text{crs}_{\text{BARG}}^{(n')})_{n' \in [\lambda]}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}$, and randomness $r := (j_1, \dots, j_N)$ for Preprocess.

We now show that [Construction 7.3](#) is both correct and succinct.

Lemma 7.4 (Correctness). *Suppose that Π_{ABO} is correct ([Definition 3.3](#)), Π_{FBH} is correct ([Definition 3.5](#)), and Π_{BARG} is complete ([Definition 5.2](#)). Then the puncturable public-coin threshold signature scheme Π_{PPCTS} of [Construction 7.3](#) satisfies signing correctness and aggregation correctness.*

Proof. Signing correctness. Fix any $\lambda \in \mathbb{N}$, any crs in the support of $\text{Setup}(1^\lambda)$, and any $m \in \mathcal{M}$. Let $(\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda, \text{crs})$, so $\text{vk} = \text{vk}^{\text{ABO}}$ and $\text{sk} = \text{sk}^{\text{ABO}}$ for some $(\text{sk}^{\text{ABO}}, \text{vk}^{\text{ABO}})$ in the support of $\text{ABO.Setup}(1^\lambda)$, and let $\sigma \leftarrow \text{Sign}(\text{sk}, m) = \text{ABO.Sign}(\text{sk}^{\text{ABO}}, m)$. Correctness of Π_{ABO} gives $\text{ABO.Verify}(\text{vk}^{\text{ABO}}, m, \sigma) = 1$, that is, $\text{PartialVerify}(\text{vk}, m, \sigma) = 1$.

Aggregation correctness. Fix any $\lambda \in \mathbb{N}$, any $k \in [2^\lambda]$, any message $m \in \mathcal{M}$, any threshold $t \in [k]$, any $T \subseteq [k]$ with $|T| \geq t$, any crs in the support of $\text{Setup}(1^\lambda)$, any $\{(\text{vk}_\ell, \text{sk}_\ell)\}_{\ell \in [k]}$ where $(\text{vk}_\ell, \text{sk}_\ell)$ is in the support of $\text{KeyGen}(1^\lambda, \text{crs})$ for each $\ell \in [k]$, and any signatures $\{\sigma_\ell\}_{\ell \in T}$ with $\text{PartialVerify}(\text{vk}_\ell, m, \sigma_\ell) = 1$ for each $\ell \in T$.

Suppose first that $k = 1$, so that $t = 1$ and $T = \{1\}$. Then Preprocess outputs $\text{vk}_{\text{agg}} = \text{vk}_1$ and $\text{ak} = \perp$, Aggregate outputs $\sigma_{\text{agg}} = \sigma_1$, and Verify takes its fallback branch, returning $\text{PartialVerify}(\text{vk}_1, m, \sigma_1) = 1$ by hypothesis. Assume $k \geq 2$ from here on.

Let $(\text{vk}_{\text{agg}}, \text{ak}) \leftarrow \text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k))$ and let $n, N = 2^n$, $\text{hk} = \text{hk}^{(n)}$ and $\text{crs}_{\text{BARG}} = \text{crs}_{\text{BARG}}^{(n)}$ be as selected there, so that hk is in the support of $\text{FBH.Setup}(1^\lambda, N, 1^{s_g^{(n)}})$ and crs_{BARG} is in the support of

$$\text{BARG.Gen}(1^\lambda, N, 1^{s_c^{(n)}}, 1^{s_t^{(n)}}).$$

Preprocessing additionally sets $\text{vk}_\ell := \text{vk}_\perp$ for each $\ell \in [k+1, N]$, samples a permutation $(j_1, \dots, j_N)$ of $[N]$, and computes $(\text{dig}, \rho_1, \dots, \rho_N) = \text{FBH.Hash}(\text{hk}, (x_1, \dots, x_N))$ where $x_{j_\ell} = (j_\ell, \text{vk}_\ell)$ for every $\ell \in [N]$, so that $\text{vk}_{\text{agg}} = (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}})$ and $\text{ak} = (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}}, \{(\text{vk}_\ell, j_\ell, \rho_{j_\ell})\}_{\ell \in [k]})$. Let $\sigma_{\text{agg}} \leftarrow \text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T})$, which forms $C := C[n, \text{hk}, \text{dig}, m]$ and $S_{\text{BARG}} := \{j_\ell\}_{\ell \in T}$, sets $w_{j_\ell} := (\text{vk}_\ell, \rho_{j_\ell}, \sigma_\ell)$ for each $\ell \in T$, and outputs $\pi_{\text{BARG}} \leftarrow \text{BARG.Prove}(\text{crs}_{\text{BARG}}, C, S_{\text{BARG}}, (w_j)_{j \in S_{\text{BARG}}})$.

Every selected instance is satisfied. Fix $\ell \in T$. Since $(j_1, \dots, j_N)$ is a permutation of $[N]$, $x_{j_\ell} = (j_\ell, \text{vk}_\ell)$ is exactly the j_ℓ^{th} block that FBH.Hash hashed, so correctness of Π_{FBH} gives $\text{FBH.Verify}(\text{hk}, \text{dig}, j_\ell, (j_\ell, \text{vk}_\ell), \rho_{j_\ell}) = 1$. Moreover, since $\ell \in T \subseteq [k]$, the key vk_ℓ is one of the keys given in the hypothesis, so $\text{ABO.Verify}(\text{vk}_\ell, m, \sigma_\ell) = \text{PartialVerify}(\text{vk}_\ell, m, \sigma_\ell) = 1$. Hence $C(j_\ell, w_{j_\ell}) = 1$.

The selected set is large enough. Since $(j_1, \dots, j_N)$ is a permutation, the map $\ell \mapsto j_\ell$ is injective, so $|S_{\text{BARG}}| = |T| \geq t$; moreover $t \leq k \leq N$, so t is a legal threshold for the batch size N .

The BARG proof verifies. The circuit C has size at most $s_c^{(n)}$. Applying completeness of Π_{BARG} to the set S_{BARG} (of size at least t) and the witnesses $(w_j)_{j \in S_{\text{BARG}}}$, each satisfying $C(j, w_j) = 1$, yields $\text{BARG.Verify}(\text{crs}_{\text{BARG}}, C, t, \pi_{\text{BARG}}) = 1$, that is, $\text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}}) = 1$. $\square$

Lemma 7.5 (Succinctness). *Suppose that Π_{ABO} runs in polynomial time, that Π_{FBH} satisfies succinctness ([Definition 3.5](#)), that Π_{BARG} satisfies succinctness ([Definition 5.2](#)), and that $s_g^{(n)}, s_t^{(n)} \leq \text{poly}(\lambda, n)$ for all $n \in [\lambda]$. Then the puncturable public-coin threshold signature scheme Π_{PPCTS} of [Construction 7.3](#) is succinct: there is a fixed polynomial $\text{poly}(\cdot, \cdot)$ such that for all $\lambda \in \mathbb{N}$, all $k \in [2^\lambda]$, all crs in the support of $\text{Setup}(1^\lambda)$, all messages $m \in \mathcal{M}$, all $t \in [k]$, all $T \subseteq [k]$, all $(\text{vk}_\ell, \text{sk}_\ell)$ in the support of $\text{KeyGen}(1^\lambda, \text{crs})$ for $\ell \in [k]$, all $(\text{vk}_{\text{agg}}, \text{ak})$ in the support of $\text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k))$, and all σ_{agg} in the support of $\text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T})$ for signatures σ_ℓ , we have $|\text{vk}_{\text{agg}}| \leq \text{poly}(\lambda, \log k)$ and $|\sigma_{\text{agg}}| \leq \text{poly}(\lambda, \log k)$.*

Proof. For $k = 1$ we have $\text{vk}_{\text{agg}} = \text{vk}_1$ and $\sigma_{\text{agg}} \in \{\sigma_1, \perp\}$, of length at most $\max(\ell_{\text{vk}_{\text{ABO}}}, \ell_{\text{ABO}}) \leq \text{poly}(\lambda)$ since Π_{ABO} runs in polynomial time, so assume $k \geq 2$. Write $n, N = 2^n$, $\text{hk} = \text{hk}^{(n)}$ and $\text{crs}_{\text{BARG}} = \text{crs}_{\text{BARG}}^{(n)}$ for the exponent and the pair that Preprocess selects, and recall $2^{n-1} < k \leq N$, so that $\log N = n \leq \log k + 1$ and n itself is described by $O(\log \lambda)$ bits.

Hash key, digest, and openings. By succinctness of Π_{FBH} , $|\text{hk}|, |\text{dig}|, \ell_\rho \leq \text{poly}(\lambda, \log N, s_g^{(n)}) \leq \text{poly}(\lambda, \log k)$ using the hypothesis on $s_g^{(n)}$.

Circuit size. The circuit $C[n, \text{hk}, \text{dig}, m]$ of Figure 5 hardwires $(n, \text{hk}, \text{dig}, m)$, of total length $\text{poly}(\lambda, \log k)$ by the previous paragraph and since $m \in \mathcal{M}$ has length $\text{poly}(\lambda)$. On an index $j \in [N]$, described by n bits, and a witness $(\text{vk}, \rho, \sigma)$ of length $\ell_w = \ell_{\text{vk}_{\text{ABO}}} + \ell_\rho + \ell_{\text{ABO}} \leq \text{poly}(\lambda, \log k)$ (since $\ell_{\text{vk}_{\text{ABO}}}, \ell_{\text{ABO}} = \text{poly}(\lambda)$, as Π_{ABO} runs in polynomial time, and $\ell_\rho \leq \text{poly}(\lambda, \log k)$ by the previous paragraph), it runs FBH.Verify and ABO.Verify , both efficient. Hence the circuit has size at most $\text{poly}(\lambda, n) \leq \text{poly}(\lambda, \log k)$; we take $s_c^{(n)}$ to be this bound for batch size 2^n , as Setup requires. For the same reason the hypotheses on $s_g^{(n)}$ and $s_t^{(n)}$ are exactly what Construction 7.3 needs of them.

CRS and verification key size. By the compact CRS property of Π_{BARG} , and since crs_{BARG} is in the support of $\text{BARG.Gen}(1^\lambda, N, 1^{s_c^{(n)}}, 1^{s_t^{(n)}})$, we have $|\text{crs}_{\text{BARG}}| \leq \text{poly}(\lambda, \log N, s_c^{(n)}, s_t^{(n)}) \leq \text{poly}(\lambda, \log k)$ using the previous paragraph and the hypothesis on $s_t^{(n)}$. Combined with $|n|, |\text{hk}|, |\text{dig}| \leq \text{poly}(\lambda, \log k)$, this gives $|\text{vk}_{\text{agg}}| \leq \text{poly}(\lambda, \log k)$.

Aggregate signature size. An aggregate signature σ_{agg} in the support of $\text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T})$ is a proof π_{BARG} in the support of $\text{BARG.Prove}(\text{crs}_{\text{BARG}}, C, S_{\text{BARG}}, (w_j)_{j \in S_{\text{BARG}}})$ for the circuit C of the previous paragraph and the same crs_{BARG} ; by the succinct-proof property of Π_{BARG} , $|\pi_{\text{BARG}}| \leq \text{poly}(\lambda, \log N, s_c^{(n)}, s_t^{(n)}) \leq \text{poly}(\lambda, \log k)$.

Note that crs itself is not required to be short, and indeed it is not: it holds λ hash keys and λ BARG common reference strings, one per possible batch size. $\square$

Theorem 7.6 (Punctured Security). *Assume that Π_{PFW} is correct (Definition 6.3), Π_{FBH} satisfies perfect function binding (Definition 3.5), Π_{ABO} satisfies punctured correctness (Definition 3.3) and has the special value $\text{vk}_\perp$ assumed in Construction 7.3, and Π_{BARG} is statistically sound (Definition 5.2). Then Π_{PPTS} of Construction 7.3, with Sim as defined there, satisfies punctured static security.*

Proof. Let $\mathcal{A}$ be a (potentially unbounded) adversary for the punctured static security game of Definition 7.1. We first dispose of the runs on which $\mathcal{A}$ outputs $k = 1$. There the constraints $t \in [k]$ and $t > |S|$ force $t = 1$ and $S = \emptyset$, so Sim takes its $k = 1$ branch and party 1 receives the punctured key vk_1 of $\text{ABO.SetupPunc}(1^\lambda, m^*)$, and Preprocess returns $\text{vk}_{\text{agg}} = \text{vk}_1$. Whatever forgery $\mathcal{A}$ outputs, $b = \text{PartialVerify}(\text{vk}_1, m^*, \sigma_{\text{agg}}) = \text{ABO.Verify}(\text{vk}_1, m^*, \sigma_{\text{agg}}) = 0$ by punctured correctness of Π_{ABO} . Hence $\Pr[b = 1]$ is unchanged if we restrict attention to the runs with $k \geq 2$, and the adversary below aborts on the others, which only lowers its winning probability. We build the following (potentially unbounded) adversary $\mathcal{B}$ for the statistical soundness game of Definition 5.2.

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain $1^k, S \subseteq [k]$, and $m^* \in \mathcal{M}$, and aborts if $k = 1$.
2. Algorithm $\mathcal{B}$ lets $n \in [\lambda]$ be the unique exponent with $k \in [2^{n-1} + 1, 2^n]$, and sets $N := 2^n, \bar{S} := [N] \setminus S$ and $c := |S|$. It samples $K \leftarrow \text{PFW.Setup}(1^\lambda, N, c)$ and $K_{\text{PRF}} \leftarrow \text{PRF.Setup}(1^\lambda, N, 1^{c'})$, and samples $(j_1, \dots, j_N) \leftarrow \text{Permute}^*(1^N, S)$ with the string x inside Permute^* set to $x_j := \text{PFW.Query}(K, j)$ for each $j \in [N]$.
3. Algorithm $\mathcal{B}$ sends the batch size $N, 1^{s_c^{(n)}}, 1^{s_t^{(n)}}$, and the description of $\text{Ext} := \text{PFW.Query}(K, \cdot)$ to the challenger, which is admissible because Ext has description size at most $s_t^{(n)}$, and receives crs_{BARG} .
4. For each $\ell \in \bar{S} \cap [k]$, algorithm $\mathcal{B}$ sets $(\text{sk}_\ell, \text{vk}_\ell) := \text{ABO.SetupPunc}(1^\lambda, m^*, \text{PRF.Eval}(K_{\text{PRF}}, j_\ell))$.
5. Algorithm $\mathcal{B}$ samples $\text{hk} := \text{hk}^{(n)} \leftarrow \text{FBH.SetupBinding}(1^\lambda, N, 1^{s_g^{(n)}}, g)$ for the predicate $g := g[K, K_{\text{PRF}}, m^*]$ of Figure 6, sets $\text{crs}_{\text{BARG}}^{(n)} := \text{crs}_{\text{BARG}}$, and for each $n' \in [\lambda] \setminus \{n\}$ samples $\text{hk}^{(n')} \leftarrow \text{FBH.Setup}(1^\lambda, 2^{n'}, 1^{s_g^{(n')}})$ and $\text{crs}_{\text{BARG}}^{(n')} \leftarrow \text{BARG.Gen}(1^\lambda, 2^{n'}, 1^{s_c^{(n')}}), 1^{s_t^{(n')}})$.
6. Algorithm $\mathcal{B}$ sets $\text{crs} := (\text{hk}^{(n')}, \text{crs}_{\text{BARG}}^{(n')})_{n' \in [\lambda]}$ and sends $\text{crs}, (\text{vk}_\ell)_{\ell \in [k] \setminus S}$, and $r := (j_1, \dots, j_N)$ to $\mathcal{A}$.
7. When $\mathcal{A}$ makes a signing query (ℓ, m) for $\ell \in [k] \setminus S$ and $m \neq m^*$, algorithm $\mathcal{B}$ responds with $\text{Sign}(\text{sk}_\ell, m)$.
8. Algorithm $\mathcal{A}$ outputs arbitrary corrupted verification keys $(\text{vk}_\ell)_{\ell \in S}$, a threshold $t \in [k]$ with $t > |S|$, and a forgery σ_{agg} .
9. Algorithm $\mathcal{B}$ completes the computation of $\text{Preprocess}(\text{crs}, (\text{vk}_1, \dots, \text{vk}_k); r)$: it sets $\text{vk}_\ell := \text{vk}_\perp$ for each $\ell \in [k + 1, N]$ and hashes

$$(\text{dig}, \rho_1, \dots, \rho_N) = \text{FBH.Hash}(\text{hk}, (x_1, \dots, x_N)), \quad \text{where } x_{j_\ell} := (j_\ell, \text{vk}_\ell) \text{ for each } \ell \in [N].$$

10. Algorithm $\mathcal{B}$ sets $\text{vk}_{\text{agg}} := (n, \text{hk}, \text{dig}, \text{crs}_{\text{BARG}})$ and $C^* := C[n, \text{hk}, \text{dig}, m^*]$ from Figure 5, and outputs $(C^*, t, \sigma_{\text{agg}})$ to the BARG challenger; this is admissible because $t \leq k \leq N$.

Algorithm $\mathcal{B}$ perfectly simulates the punctured static security game for $\mathcal{A}$: it samples K, K_{PRF} , the permutation, the honest parties' keys $((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}$ via ABO.SetupPunc , the binding key $\text{hk}^{(n)}$ and the $\lambda - 1$ remaining pairs of crs exactly as Sim does, and the one value it does not sample itself, crs_{BARG} , is returned by its challenger distributed as $\text{BARG.SoundGen}(1^\lambda, N, 1^{s_c^{(n)}}, 1^{s_t^{(n)}}, \text{Ext})$, which is exactly the distribution step 8 of Sim would have used. Steps 9 and 10 then run Preprocess on the simulated honest verification keys, the corrupted verification keys supplied by $\mathcal{A}$ in step 8, and the very randomness r that $\mathcal{B}$ handed $\mathcal{A}$, as the challenger of the punctured static security game does. In particular, $\mathcal{A}$'s output $((\text{vk}_\ell)_{\ell \in S}, t, \sigma_{\text{agg}})$ has the same distribution as in the real game, and $b = \text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}}) = \text{BARG.Verify}(\text{crs}_{\text{BARG}}, C^*, t, \sigma_{\text{agg}})$.

We claim that whenever $b = 1$, the tuple $(C^*, t, \sigma_{\text{agg}})$ that $\mathcal{B}$ outputs also satisfies the two remaining winning conditions of the statistical soundness game, namely that $\text{Ext} := \text{PFW.Query}(K, \cdot)$ is a trapdoor for the batch language of C^* (Section 5) and that $\text{Ext}(\text{root}) < t$. Since $\mathcal{B}$'s challenger already fixes Ext before sampling crs_{BARG} , this holds deterministically, for every choice of K, K_{PRF} , and $(j_1, \dots, j_N)$, rather than with some probability of failure.

Validity and propagation of Ext hold for every K in the support of $\text{PFW.Setup}(1^\lambda, N, c)$, by correctness of Π_{PFW} .

Before turning to leaf soundness, we record that the vector of blocks $(x_1, \dots, x_N)$ hashed in step 9 satisfies g at every slot, which is the hypothesis under which perfect function binding of Π_{FBH} speaks. Indeed, fix $\ell \in [N]$ and consider the block $x_{j_\ell} = (j_\ell, \text{vk}_\ell)$. If $\ell \in S$ then $j_\ell \in \mathcal{I} := \{j \in [N] : \text{PFW.Query}(K, j) = 1\} = \{j_{\ell'}\}_{\ell' \in S}$, so $\text{PFW.Query}(K, j_\ell) = 1$ and the first of the three disjuncts of g holds for every adversarial choice of vk_ℓ , even one depending on crs , the honest verification keys, r , and the signing responses. If $\ell \in \bar{S} \cap [k]$ then vk_ℓ is by construction the verification key output by $\text{ABO.SetupPunc}(1^\lambda, m^*; \text{PRF.Eval}(K_{\text{PRF}}, j_\ell))$, which is the third disjunct. And if $\ell \in [k+1, N]$ then $\text{vk}_\ell = \text{vk}_\perp$, which is the second.

For leaf soundness, fix a leaf $j \in [N]$ with $\text{Ext}(j) = 0$, that is, $j \notin \mathcal{I}$, and suppose toward a contradiction that some witness $w = (\text{vk}, \rho, \sigma)$ satisfies $C^*(j, w) = 1$. The first check of C^* gives $\text{FBH.Verify}(\text{hk}, \text{dig}, j, (j, \text{vk}), \rho) = 1$. Since $\text{hk} \leftarrow \text{FBH.SetupBinding}(1^\lambda, N, 1^{s_g^{(n)}})$, g binds on the conjunction $\mathbf{x} \mapsto \bigwedge_{i \in [N]} g(x_i)$, and $(\text{dig}, \rho_1, \dots, \rho_N)$ is the hash of the vector of blocks of the previous paragraph, perfect function binding of Π_{FBH} gives $g((j, \text{vk})) = 1$; this holds for every accepting opening at slot j , not just for the block actually hashed there. Since $j \in [N]$, the range case of g does not fire; and since $\text{Ext}(j) = \text{PFW.Query}(K, j) = 0$, its first disjunct fails as well, so either $\text{vk} = \text{vk}_\perp$ or vk is the verification key output by $\text{ABO.SetupPunc}(1^\lambda, m^*; \text{PRF.Eval}(K_{\text{PRF}}, j))$. In the first case, $\text{ABO.Verify}(\text{vk}_\perp, m^*, \sigma') = 0$ for every σ' by the assumed property of $\text{vk}_\perp$. In the second case, vk is in the support of $\text{ABO.SetupPunc}(1^\lambda, m^*)$, so punctured correctness of Π_{ABO} gives $\text{ABO.Verify}(\text{vk}, m^*, \sigma') = 0$ for every σ' . Either way this contradicts the second check of C^* , which requires $\text{ABO.Verify}(\text{vk}, m^*, \sigma) = 1$. So the j^{th} instance of C^* is a no instance, as required.

Hence Ext is a trapdoor for the batch language of C^* , and $\text{Ext}(\text{root}) = \sum_{j \in [N]} \text{Ext}(j) = |\mathcal{I}| = c = |S|$ by the correct-weight property of Π_{PFW} ; since $t > |S|$ (as $\mathcal{A}$'s output must satisfy in the punctured static security game), this gives $\text{Ext}(\text{root}) < t$.

Therefore $\mathcal{B}$ wins the statistical soundness game exactly when $b = 1$, which as noted entails $k \geq 2$ and so never falls on a run where $\mathcal{B}$ aborts; hence $\Pr[b = 1]$ equals $\mathcal{B}$'s winning probability, and statistical soundness of Π_{BARG} gives a negligible function $\text{negl}(\cdot)$ with $\Pr[b = 1] \leq \text{negl}(\lambda)$ for all $\lambda \in \mathbb{N}$, as required. $\square$

Theorem 7.7 (Mode Indistinguishability). *Assume that Π_{PFW} is correct and pseudorandom (Definition 6.3), Π_{PRF} is a secure PRF (Definition 3.1), Π_{FBH} satisfies mode indistinguishability (Definition 3.5), Π_{BARG} satisfies mode indistinguishability (Definition 5.2), and Π_{ABO} satisfies verification key indistinguishability (Definition 3.3). Then Π_{PPCTS} of Construction 7.3, with Sim as defined there, satisfies mode indistinguishability.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game against Construction 7.3. Note that this game never invokes Preprocess : the challenger sends $\mathcal{A}$ only crs , the honest parties' verification keys, and the randomness r , and answers signing queries, so the following hybrids need not track a digest at all. The adversary may choose corrupted keys locally; none are generated or sent by the challenger. Throughout, once $\mathcal{A}$ has output 1^k and S with $k \geq 2$, we write $n \in [\lambda]$ for the unique exponent with $k \in [2^{n-1} + 1, 2^n]$, and $N := 2^n$, $\bar{S} := [N] \setminus S$ and $c := |S|$, exactly as Sim does; the case $k = 1$ is treated after the hybrids are defined. For a bit $b \in \{0, 1\}$ we define a sequence of hybrid experiments; each is described as a modification of the previous one, and all of them output $\mathcal{A}$'s guess b' .

- Hyb_0 : the mode indistinguishability game with $b = 0$. In full:

1. Algorithm $\mathcal{A}(1^\lambda)$ outputs 1^k with $k \in [2, 2^\lambda]$, a set $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$.
2. For each $n' \in [\lambda]$, the challenger samples

$$\begin{aligned} \text{hk}^{(n')} &\leftarrow \text{FBH.Setup}(1^\lambda, 2^{n'}, 1^{s_g^{(n')}}) \\ \text{crs}_{\text{BARG}}^{(n')} &\leftarrow \text{BARG.Gen}(1^\lambda, 2^{n'}, 1^{s_c^{(n')}}), \end{aligned}$$

and sets $\text{crs} = (\text{hk}^{(n')}, \text{crs}_{\text{BARG}}^{(n')})_{n' \in [\lambda]}$.

3. The challenger samples a uniformly random permutation $r = (j_1, \dots, j_N)$ of $[N]$.
 4. For each $\ell \in [k] \setminus S$, the challenger samples $(\text{sk}_\ell, \text{vk}_\ell) \leftarrow \text{ABO.Setup}(1^\lambda)$.
 5. The challenger sends crs , $(\text{vk}_\ell)_{\ell \in [k] \setminus S}$, and r to $\mathcal{A}$.
 6. Throughout the remainder of the experiment, $\mathcal{A}$ may adaptively make signing queries: on a query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, the challenger replies with $\text{Sign}(\text{sk}_\ell, m)$.
 7. Algorithm $\mathcal{A}$ outputs b' , the output of the experiment.
- Hyb_1 : same as Hyb_0 , except that the permutation is sampled using the alternative procedure. Namely:
 3. The challenger samples $r = (j_1, \dots, j_N) \leftarrow \text{Permute}^*(1^N, S)$ of Figure 4, where S is the corrupted set, of size c .
 - Hyb_2 : same as Hyb_1 , except that x is sampled using the pseudorandom fixed-weight string generator. Namely, before step 2 the challenger additionally samples $K \leftarrow \text{PFW.Setup}(1^\lambda, N, c)$, and step 3 becomes:
 3. The challenger samples $r = (j_1, \dots, j_N) \leftarrow \text{Permute}^*(1^N, S)$, where the string x inside Permute^* is set as $x_j = \text{PFW.Query}(K, j)$.
 - Hyb_3 : same as Hyb_2 , except that the keys of the honest parties $[k] \setminus S$ are punctured at m^* . Namely:
 4. For each $\ell \in [k] \setminus S$, the challenger samples $r'_\ell \xleftarrow{R} \{0, 1\}^{\ell_{r'}}$ and sets $(\text{sk}_\ell, \text{vk}_\ell) \leftarrow \text{ABO.SetupPunc}(1^\lambda, m^*; r'_\ell)$.

Since $\mathcal{A}$ never receives sk_ℓ for any $\ell \in [k] \setminus S$, it never sees a punctured signing key directly.
 - Hyb_4 : same as Hyb_3 , except that the randomness of the punctured setups is derived from a PRF. Namely, before step 2 the challenger additionally samples $K_{\text{PRF}} \leftarrow \text{PRF.Setup}(1^\lambda, N, 1^{\ell_{r'}})$, and step 4 becomes:
 4. For each $\ell \in [k] \setminus S$, the challenger sets $(\text{sk}_\ell, \text{vk}_\ell) \leftarrow \text{ABO.SetupPunc}(1^\lambda, m^*; \text{PRF.Eval}(K_{\text{PRF}}, j_\ell))$.
 - Hyb_5 : same as Hyb_4 , except that the hash key of the instance actually used binds on the predicate $g := g[K, K_{\text{PRF}}, m^*]$ of Figure 6. Namely:
 2. For each $n' \in [\lambda]$, the challenger samples $\text{crs}_{\text{BARG}}^{(n')} \leftarrow \text{BARG.Gen}(1^\lambda, 2^{n'}, 1^{s_c^{(n')}})$, and samples $\text{hk}^{(n')} \leftarrow \text{FBH.Setup}(1^\lambda, 2^{n'}, 1^{s_g^{(n')}})$ for each $n' \neq n$ and $\text{hk}^{(n)} \leftarrow \text{FBH.SetupBinding}(1^\lambda, N, 1^{s_g^{(n)}})$. It sets $\text{crs} = (\text{hk}^{(n')}, \text{crs}_{\text{BARG}}^{(n')})_{n' \in [\lambda]}$.
 - Hyb_6 : same as Hyb_5 , except that the BARG of that same instance is now in sound mode. Namely:
 2. For each $n' \in [\lambda]$, the challenger samples $\text{hk}^{(n')}$ as in Hyb_5 , samples $\text{crs}_{\text{BARG}}^{(n')} \leftarrow \text{BARG.Gen}(1^\lambda, 2^{n'}, 1^{s_c^{(n')}})$ for each $n' \neq n$ and $\text{crs}_{\text{BARG}}^{(n)} \leftarrow \text{BARG.SoundGen}(1^\lambda, N, 1^{s_c^{(n)}})$, where $\text{Ext}(v) = \text{PFW.Query}(K, v)$ for every $v \in [2N - 1]$. It sets $\text{crs} = (\text{hk}^{(n')}, \text{crs}_{\text{BARG}}^{(n')})_{n' \in [\lambda]}$.

On the runs where $\mathcal{A}$ outputs $k = 1$ every one of these experiments degenerates: Preprocess consumes no randomness, so each of them sets $r := \perp$ and samples no K , no K_{PRF} and no permutation, and each leaves all λ instances of crs in normal mode, since the instance n that Hyb_5 and Hyb_6 switch is not defined. The only surviving difference is the puncturing of party 1's key when $S = \emptyset$, so there $\text{Hyb}_0 = \text{Hyb}_1 = \text{Hyb}_2$ and $\text{Hyb}_3 = \text{Hyb}_4 = \text{Hyb}_5 = \text{Hyb}_6$. The claims below cover this case: all but [Claim 7.10](#) are equalities on such runs, and the reduction of that claim skips its sampling of K and of the permutation.

Comparing with $\text{Sim}(1^\lambda, 1^k, S, m^*)$ ([Construction 7.3](#)) step by step, Hyb_6 samples $K, K_{\text{PRF}}, (j_1, \dots, j_N)$, the honest parties' punctured key pairs, the λ hash keys, and the λ BARG common reference strings exactly as Sim does. It sends $\mathcal{A}$ the same public view as the simulated-mode challenger: $\text{crs}, (\text{vk}_\ell)_{\ell \in [k] \setminus S}$, and $r = (j_1, \dots, j_N)$, retaining the honest signing keys to answer signing queries. On the runs with $k = 1$ the same holds against the $k = 1$ branch of Sim , which likewise samples $\text{crs} \leftarrow \text{Setup}(1^\lambda)$, samples a punctured key for party 1 if $S = \emptyset$ and no keys if $S = \{1\}$, and returns $r = \perp$. So Hyb_6 is exactly the mode indistinguishability game with $b = 1$.

Claim 7.8. *For any adversary $\mathcal{A}$, we have $\Pr[\text{Hyb}_0(\mathcal{A}) = 1] = \Pr[\text{Hyb}_1(\mathcal{A}) = 1]$.*

Proof. The two experiments differ only in how the permutation $r = (j_1, \dots, j_N)$ is sampled. By [Proposition 6.11](#), applied with the parameters N and S , the output of $\text{Permute}^*(1^N, S)$ is a uniformly random permutation of $[N]$, exactly as in Hyb_0 . As nothing else changes, the two experiments are identically distributed. $\square$

Claim 7.9. *Assume Π_{PFW} is pseudorandom. Then for any efficient $\mathcal{A}$ there is a negligible negl such that*

$$|\Pr[\text{Hyb}_1(\mathcal{A}) = 1] - \Pr[\text{Hyb}_2(\mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

Proof. The experiments differ only in whether the string x inside Permute^* is a uniform weight- c string (Hyb_1) or $x_j = \text{PFW.Query}(K, j)$ for $K \leftarrow \text{PFW.Setup}(1^\lambda, N, c)$ (Hyb_2). We build $\mathcal{B}$ against Π_{PFW} 's pseudorandomness: $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain $1^k, S, m^*$, computes N and $c = |S|$, sends the power of two N and the weight c to the pseudorandomness challenger, and receives a string x . It completes the experiment as in $\text{Hyb}_1/\text{Hyb}_2$ using x inside Permute^* (all other steps are identical in the two hybrids and do not depend on x), and outputs whatever $\mathcal{A}$ outputs. If x is uniform of weight c ($b_{\text{PFW}} = 0$), $\mathcal{B}$ simulates Hyb_1 ; if $x_j = \text{PFW.Query}(K, j)$ ($b_{\text{PFW}} = 1$), it simulates Hyb_2 . Thus $\mathcal{B}$'s advantage equals the claimed difference, which is negligible. $\square$

Claim 7.10. *Assume Π_{ABO} satisfies verification key indistinguishability. Then for any efficient $\mathcal{A}$ there is a negligible negl such that*

$$|\Pr[\text{Hyb}_2(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

Proof. Since $\mathcal{A}$ is efficient and writes the number of parties in unary, there is a polynomial $k_{\max} = k_{\max}(\lambda)$, depending only on $\mathcal{A}$, such that every first message of $\mathcal{A}$ satisfies $k \leq k_{\max}$; in particular the number $c' := k - |S|$ of honest parties satisfies $c' \leq k_{\max}$. Write $[k] \setminus S = \{\ell_1 < \dots < \ell_{c'}\}$.

For $i \in [0, k_{\max}]$, let $\text{Hyb}_{2,i}$ be Hyb_2 except that step 4 samples $(\text{sk}_{\ell_{i'}}, \text{vk}_{\ell_{i'}}) \leftarrow \text{ABO.SetupPunc}(1^\lambda, m^*)$ for every $i' \leq \min(i, c')$, and $(\text{sk}_\ell, \text{vk}_\ell) \leftarrow \text{ABO.Setup}(1^\lambda)$ for every other honest party $\ell \in [k] \setminus S$. We have $\text{Hyb}_{2,0} = \text{Hyb}_2$, and $\text{Hyb}_{2,i} = \text{Hyb}_3$ for every $i \geq c'$; in particular $\text{Hyb}_{2,k_{\max}} = \text{Hyb}_3$.

Consider the following adversary $\mathcal{B}$ for the verification key indistinguishability game of [Definition 3.3](#).

1. Sample $i \xleftarrow{R} [k_{\max}]$.
2. Run the first stage of $\mathcal{A}(1^\lambda)$ to obtain $1^k, S$, and m^* .
3. Send $\text{msg}^* = m^*$ to the challenger and receive vk .
4. Compute N and $c = |S|$, sample $K \leftarrow \text{PFW.Setup}(1^\lambda, N, c)$, and sample $(j_1, \dots, j_N) \leftarrow \text{Permute}^*(1^N, S)$ with $x_j = \text{PFW.Query}(K, j)$, as in Hyb_2 ; write $[k] \setminus S = \{\ell_1 < \dots < \ell_{c'}\}$.
5. If $i > c'$, complete the experiment Hyb_3 on its own with this m^* , discarding vk : sample the rest of the setup, answer $\mathcal{A}$'s signing queries directly, answer the challenge as in Hyb_3 , and output $\mathcal{A}$'s guess. Otherwise, continue with the following steps.

6. Run the setup of $\text{Hyb}_{2,i-1}$ with this K , except that for the party ℓ_i it sets $\text{vk}_{\ell_i} := \text{vk}$ and samples no signing key.
7. Send $\text{crs}, (\text{vk}_{\ell})_{\ell \in [k] \setminus S}$, and $r = (j_1, \dots, j_N)$ to $\mathcal{A}$; no signing keys are revealed.
8. On a signing query (ℓ, m) with $\ell \neq \ell_i$, answer using the sampled sk_{ℓ} . On a query (ℓ_i, m) (necessarily with $m \neq m^*$), query the challenge signing oracle on m to obtain σ , and return σ .
9. Output $\mathcal{A}$'s guess.

The queries $\mathcal{B}$ forwards to the challenge signing oracle always satisfy $m \neq m^* = \text{msg}^*$, since the mode indistinguishability game restricts $\mathcal{A}$'s own signing queries to $m \neq m^*$. Beyond the signing oracle, $\mathcal{B}$ needs no signing key for ℓ_i : that party enters the rest of the experiment only through vk_{ℓ_i} , which $\mathcal{B}$ sends to $\mathcal{A}$ directly. Fix a value $i \in [k_{\max}]$ of the sampled index. On the runs with $i > c'$, the two experiments $\text{Hyb}_{2,i-1}$ and $\text{Hyb}_{2,i}$ both coincide with Hyb_3 , which $\mathcal{B}$ simulates on its own without using vk . On the runs with $i \leq c'$, if the challenger's bit is 0 then vk is distributed as the verification key of $\text{ABO.Setup}(1^\lambda)$ and $\mathcal{B}$ simulates $\text{Hyb}_{2,i-1}$, and if it is 1 then vk is distributed as that of $\text{ABO.SetupPunc}(1^\lambda, m^*)$ and $\mathcal{B}$ simulates $\text{Hyb}_{2,i}$. In both cases, conditioned on the sampled index being i , algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\text{Hyb}_{2,i-1}(\mathcal{A}) = 1]$ when the challenger's bit is 0 and with probability $\Pr[\text{Hyb}_{2,i}(\mathcal{A}) = 1]$ when it is 1.

We bound the signed sum over the inner hybrids rather than each term separately: applying the guarantee to each i would yield $k_{\max}$ negligible functions, and $k_{\max}$ grows with λ , so their maximum need not be negligible. Since i is sampled uniformly from $[k_{\max}]$ and independently of the challenger's bit, verification key indistinguishability gives a single negligible negl' with

$$\begin{aligned} \text{negl}'(\lambda) &\geq \left| \frac{1}{k_{\max}} \sum_{i \in [k_{\max}]} \left(\Pr[\text{Hyb}_{2,i-1}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{2,i}(\mathcal{A}) = 1] \right) \right| \\ &= \frac{1}{k_{\max}} \left| \Pr[\text{Hyb}_2(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1] \right|, \end{aligned}$$

and the claim follows with $\text{negl} = k_{\max} \cdot \text{negl}'$, which is negligible because $k_{\max} \leq \text{poly}(\lambda)$. $\square$

Claim 7.11. Assume Π_{PRF} is a secure PRF. Then for any efficient $\mathcal{A}$ there is a negligible negl such that

$$\left| \Pr[\text{Hyb}_3(\mathcal{A}) = 1] - \Pr[\text{Hyb}_4(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. The experiments differ only in whether the randomness of the punctured setups is uniform (Hyb_3) or the output of $\text{PRF.Eval}(K_{\text{PRF}}, \cdot)$ (Hyb_4). We build $\mathcal{B}$ against pseudorandomness: $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain $1^k, S$, and m^* , computes n and $N = 2^n$, sends the lengths $1^n, 1^{t'}$ to the challenger, samples the rest of the setup as in the two hybrids, and for each $\ell \in [k] \setminus S$ queries its oracle on $j_\ell - 1 \in [0, N - 1]$, an n -bit input, to obtain r'_ℓ and samples $(\text{sk}_\ell, \text{vk}_\ell) \leftarrow \text{ABO.SetupPunc}(1^\lambda, m^*; r'_\ell)$. Its queries are distinct, since $\ell \mapsto j_\ell$ is a permutation, and the key K_{PRF} is needed nowhere else in the two hybrids. It completes the experiment and outputs $\mathcal{A}$'s guess. If the oracle is a random function then $\mathcal{B}$ simulates Hyb_3 , and if it is $\text{PRF.Eval}(K_{\text{PRF}}, \cdot)$ then it simulates Hyb_4 , so its advantage equals the claimed difference. $\square$

Claim 7.12. Assume Π_{FBH} satisfies mode indistinguishability. Then for any efficient $\mathcal{A}$ there is a negligible negl such that

$$\left| \Pr[\text{Hyb}_4(\mathcal{A}) = 1] - \Pr[\text{Hyb}_5(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. The experiments differ only in how $\text{hk}^{(n)}$ is sampled, which are exactly the two modes of [Definition 3.5](#). We build $\mathcal{B}$: it runs $\mathcal{A}(1^\lambda)$ to obtain $1^k, S$, and m^* , computes n and $N = 2^n$, samples K, K_{PRF} , the permutation, and the honest key pairs $((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}$ via ABO.SetupPunc as in Hyb_4 , sends $N, 1^{s_g^{(n)}}$, and the description of $g[K, K_{\text{PRF}}, m^*]$ to the challenger, and receives $\text{hk}^{(n)}$. It is admissible because g has description size at most $s_g^{(n)}$. Algorithm $\mathcal{B}$ then samples the remaining $\lambda - 1$ hash keys and all λ common reference strings $\text{crs}_{\text{BARG}}^{(n')} \leftarrow \text{BARG.Gen}(1^\lambda, 2^{n'}, 1^{s_c^{(n')}}, 1^{s_t^{(n')}})$ as in the two hybrids, assembles crs , sends $\text{crs}, (\text{vk}_\ell)_{\ell \in [k] \setminus S}$, and r to $\mathcal{A}$, answers its signing queries with the sampled keys, and outputs $\mathcal{A}$'s guess. In mode $b_{\text{FBH}} = 0$ it simulates Hyb_4 and in mode $b_{\text{FBH}} = 1$ it simulates Hyb_5 , so its advantage equals the claimed difference. $\square$

Claim 7.13. Assume Π_{BARG} satisfies mode indistinguishability. Then for any efficient $\mathcal{A}$ there is a negligible negl such that

$$|\Pr[\text{Hyb}_5(\mathcal{A}) = 1] - \Pr[\text{Hyb}_6(\mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

Proof. The experiments differ only in how $\text{crs}_{\text{BARG}}^{(n)}$ is sampled. We build $\mathcal{B}$ against BARG mode indistinguishability: it samples the whole Hyb_5 setup except $\text{crs}_{\text{BARG}}^{(n)}$ (i.e., K , K_{PRF} , the permutation, the honest key pairs, the λ hash keys, and the $\lambda - 1$ remaining BARG common reference strings), constructs $\text{Ext} = \text{PFW.Query}(K, \cdot)$ from the key K it sampled, sends $N, 1^{s_c^{(n)}}, 1^{s_t^{(n)}}$, Ext to the challenger, and receives $\text{crs}_{\text{BARG}}^{(n)}$. It completes the experiment with this CRS, sending crs , $(\text{vk}_\ell)_{\ell \in [k] \setminus S}$, and r to $\mathcal{A}$ and answering signing queries with the sampled keys, and outputs $\mathcal{A}$'s guess. Since Ext has description size at most $s_t^{(n)}$, $\mathcal{B}$ is admissible; in mode $b_{\text{BARG}} = 0$ it simulates Hyb_5 and in mode $b_{\text{BARG}} = 1$ it simulates Hyb_6 , so its advantage equals the claimed difference. $\square$

Combining Claim 7.8 through Claim 7.13, we obtain for any efficient $\mathcal{A}$ a negligible function negl such that

$$|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_6(\mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

Since Hyb_0 is exactly the mode indistinguishability game with $b = 0$ and Hyb_6 is exactly the mode indistinguishability game with $b = 1$, this establishes mode indistinguishability, completing the proof. $\square$

8 Puncturable Public-Coin Weighted Threshold Signatures

In this section, we generically upgrade a threshold signature scheme to a *weighted* threshold signature scheme. While we focus on punctured security and public-coin preprocessing, we note the transformation is generic and we expect it to work for regular security and centralized or silent setup schemes.

The idea is simple: the weights are decomposed to their bit representation, and a separate unweighted copy is run for each bit index. The verification algorithm assigns the corresponding weight to each copy, and checks that the sum indeed matches the input threshold.

Definition 8.1 (Puncturable Public-Coin Weighted Threshold Signatures). A puncturable public-coin weighted threshold signature scheme Π_{PPCWTs} for a message space $\mathcal{M}$ is a tuple of efficient algorithms

$$\Pi_{\text{PPCWTs}} = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{PartialVerify}, \text{Preprocess}, \text{Aggregate}, \text{Verify}, \text{Sim})$$

with the following syntax:

- $\text{Setup}(1^\lambda, 1^L) \rightarrow \text{crs}$: On input a security parameter λ and a bound on the bit-representation of weights L , the setup algorithm outputs a common reference string crs .
- $\text{KeyGen}(1^\lambda, 1^L, \text{crs}) \rightarrow (\text{vk}, \text{sk})$: On input a security parameter λ , a bound on the bit-representation of weights L , and a common reference string crs , the key-generation algorithm outputs a verification key vk and a signing key sk .
- $\text{Sign}(\text{sk}, m) \rightarrow \sigma$: On input a signing key sk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ .
- $\text{PartialVerify}(\text{vk}, m, \sigma) \rightarrow b$: On input a verification key vk , a message $m \in \mathcal{M}$, and a signature σ , the partial-verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.
- $\text{Preprocess}(\text{crs}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k))) \rightarrow (\text{vk}_{\text{agg}}, \text{ak})$: On input a common reference string crs and a vector of $k \in [2^\lambda]$ verification key and weight pairs (vk_ℓ, w_ℓ) such that $w_\ell \in [0, 2^L - 1]$, the preprocessing algorithm assigns the parties their weights and outputs an aggregated verification key vk_{agg} and an aggregation key ak . The length of its random tape is determined by crs and the weights.
- $\text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T}) \rightarrow \sigma_{\text{agg}}$: On input the aggregation key ak , a message $m \in \mathcal{M}$, and a set of index and signature pairs $\{(\ell, \sigma_\ell)\}_{\ell \in T}$ for $T \subseteq [k]$, the aggregation algorithm outputs an aggregate signature σ_{agg} . This algorithm is deterministic.

- $\text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}}) \rightarrow b$: On input an aggregated verification key vk_{agg} , a weight threshold $t \in [k \cdot (2^L - 1)]$, a message $m \in \mathcal{M}$, and an aggregate signature σ_{agg} , the verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.
- $\text{Sim}(1^\lambda, 1^L, 1^k, \{w_\ell\}_{\ell \in [k]}, S, m^*) \rightarrow (\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r)$: On input a security parameter λ , a bound on the bit-representation of weights L , a number of parties k , weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$, the simulator outputs a common reference string crs , a key pair $(\text{vk}_\ell, \text{sk}_\ell)$ for each honest party $\ell \in [k] \setminus S$, and randomness r for Preprocess. The challenger retains the honest signing keys to answer signing queries; the simulator generates no keys for corrupted parties.

We require that Π_{PPCWTs} satisfy the following properties:

- **Signing correctness:** For all $\lambda, L \in \mathbb{N}$, all crs in the support of $\text{Setup}(1^\lambda, 1^L)$, and all messages $m \in \mathcal{M}$,

$$\Pr \left[\text{PartialVerify}(\text{vk}, m, \sigma) = 1 : \begin{array}{l} (\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda, 1^L, \text{crs}) \\ \sigma \leftarrow \text{Sign}(\text{sk}, m) \end{array} \right] = 1.$$

- **Aggregation correctness:** For all $\lambda, L \in \mathbb{N}$, all $k \in [2^\lambda]$, all messages $m \in \mathcal{M}$, all weights $w_1, \dots, w_k \in [0, 2^L - 1]$, all thresholds $t \in [k \cdot (2^L - 1)]$, all sets $T \subseteq [k]$ with $\sum_{\ell \in T} w_\ell \geq t$, all crs in the support of $\text{Setup}(1^\lambda, 1^L)$, all $\{(\text{vk}_\ell, \text{sk}_\ell)\}_{\ell \in [k]}$ where $(\text{vk}_\ell, \text{sk}_\ell)$ is in the support of $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$ for all $\ell \in [k]$, and all signatures $\{\sigma_\ell\}_{\ell \in T}$ where $\text{PartialVerify}(\text{vk}_\ell, m, \sigma_\ell) = 1$ for all $\ell \in T$,

$$\Pr \left[\text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}}) = 1 : \begin{array}{l} (\text{vk}_{\text{agg}}, \text{ak}) \leftarrow \text{Preprocess}(\text{crs}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k))) \\ \sigma_{\text{agg}} = \text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T}) \end{array} \right] = 1.$$

- **Punctured static security:** For any (potentially unbounded) adversary $\mathcal{A}$ and polynomial $L := L(\lambda)$, we define the punctured static security game that takes as input a security parameter λ as follows:

1. On input the security parameter 1^λ , algorithm $\mathcal{A}$ outputs a number of parties 1^k , weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$.
2. The challenger samples

$$\begin{aligned} & (\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r) \\ & \leftarrow \text{Sim}(1^\lambda, 1^L, 1^k, \{w_\ell\}_{\ell \in [k]}, S, m^*). \end{aligned}$$

It gives crs together with $(\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$.

3. Throughout the game, algorithm $\mathcal{A}$ may adaptively make signing queries: on a query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, the challenger replies with $\text{Sign}(\text{sk}_\ell, m)$.
4. Algorithm $\mathcal{A}$ outputs arbitrary corrupted verification keys $(\text{vk}_\ell)_{\ell \in S}$ (not necessarily generated by KeyGen), a threshold $t \in [k \cdot (2^L - 1)]$ with $t > \sum_{\ell \in S} w_\ell$, and a forgery σ_{agg} .
5. The challenger combines the simulated honest verification keys with the adversary's corrupted verification keys and computes $(\text{vk}_{\text{agg}}, \text{ak}) = \text{Preprocess}(\text{crs}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)))$. The output of the experiment is $b = \text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}})$.

We say Π_{PPCWTs} satisfies punctured static security if for every (potentially unbounded) adversary $\mathcal{A}$ and every polynomial $L(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have $\Pr[b = 1] \leq \text{negl}(\lambda)$ in the punctured static security game defined above.

- **Public-coin mode indistinguishability:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the mode indistinguishability game as follows:

1. On input the security parameter 1^λ , algorithm $\mathcal{A}$ outputs a bound on the bit-representation of weights 1^L , a number of parties 1^k , weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$.

2. The challenger computes crs , for each honest party $\ell \in [k] \setminus S$, a key pair $(\text{vk}_\ell, \text{sk}_\ell)$, and randomness r for Preprocess as follows:
 - If $b = 0$, it samples $\text{crs} \leftarrow \text{Setup}(1^\lambda, 1^L)$, for each $\ell \in [k] \setminus S$, $(\text{vk}_\ell, \text{sk}_\ell) \leftarrow \text{KeyGen}(1^\lambda, 1^L, \text{crs})$, and r uniformly at random.
 - If $b = 1$, it samples

$$(\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r) \leftarrow \text{Sim}(1^\lambda, 1^L, 1^k, \{w_\ell\}_{\ell \in [k]}, S, m^*).$$

It gives crs together with $(\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$.

3. Throughout the game, algorithm $\mathcal{A}$ may adaptively make signing queries: on a query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, the challenger replies with $\text{Sign}(\text{sk}_\ell, m)$.
4. Algorithm $\mathcal{A}$ outputs a guess b' , which is the output of the experiment.

We say that Π_{PPCWTs} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Succinctness:** There exist universal polynomials $p_{\text{vk}}(\cdot, \cdot)$ and $p_{\text{agg}}(\cdot, \cdot)$ such that for all $\lambda, L \in \mathbb{N}$, all $k \in [2^\lambda]$, all crs in the support of $\text{Setup}(1^\lambda, 1^L)$, all $t \in [k \cdot (2^L - 1)]$, all $(\text{vk}_\ell, \text{sk}_\ell)$ in the support of $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$ for $\ell \in [k]$, all weights $w_1, \dots, w_k \in [0, 2^L - 1]$, and all $(\text{vk}_{\text{agg}}, \text{ak})$ in the support of $\text{Preprocess}(\text{crs}, ((\text{vk}_i, w_i))_{i \in [k]})$, the following hold:
 - The size of the aggregated verification key vk_{agg} is at most $p_{\text{vk}}(\lambda, \log k, L)$.
 - The size of the aggregate signatures output by $\text{Aggregate}(\text{ak}, \cdot, \cdot)$ is at most $p_{\text{agg}}(\lambda, \log k, L)$.

Taken together, these two properties imply that the running time of $\text{Verify}(\text{vk}_{\text{agg}}, \cdot, \cdot, \cdot)$ is also $\text{poly}(\lambda, \log k, L)$.

Remark 8.2 (Centralized and Silent Variants). Similar to the unweighted case, this definition implies a centralized weighted threshold signature scheme in the standard model, and a silent weighted threshold signature scheme in the random oracle model. See [Section 10](#) for more details.

8.1 A Generic Compiler from Puncturable Public-Coin Threshold Signatures

We compile any unweighted puncturable public-coin threshold signature scheme Π_{PPCTS} into a puncturable public-coin weighted threshold signature scheme by running L independent copies of Π_{PPCTS} , one per bit of the weight budget, and reducing a weight threshold to a vector of ordinary thresholds, one per bit position.

Construction 8.3 (Puncturable Public-Coin Weighted Threshold Signatures). The only ingredient we use is a puncturable public-coin threshold signature scheme Π_{PPCTS} for a message space $\mathcal{M}$ as per [Definition 7.1](#). The algorithms of Π_{PPCWTs} are defined as follows.

- $\text{Setup}(1^\lambda, 1^L)$: On input a security parameter λ and a bound L on the bit-length of the weights, the setup algorithm proceeds as follows:
 1. For each $j \in \{0, \dots, L-1\}$, sample $\text{crs}_j \leftarrow \text{PPCTS.Setup}(1^\lambda)$.
 2. Output $\text{crs} := (\text{crs}_0, \dots, \text{crs}_{L-1})$.
- $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$: On input a security parameter λ , a bound L on the bit-length of the weights, and a common reference string $\text{crs} = (\text{crs}_0, \dots, \text{crs}_{L-1})$, the key-generation algorithm proceeds as follows:
 1. For each $j \in \{0, \dots, L-1\}$, sample $(\text{vk}_j, \text{sk}_j) \leftarrow \text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$.
 2. Output $\text{vk} := (\text{vk}_0, \dots, \text{vk}_{L-1})$ and $\text{sk} := (\text{sk}_0, \dots, \text{sk}_{L-1})$.

- **Sign**(sk, m): On input a signing key $sk = (sk_0, \dots, sk_{L-1})$ and a message $m \in \mathcal{M}$, the signing algorithm proceeds as follows:
 1. For each $j \in \{0, \dots, L-1\}$, compute $\sigma_j \leftarrow \text{PPCTS.Sign}(sk_j, m)$.
 2. Output $\sigma := (\sigma_0, \dots, \sigma_{L-1})$.
- **PartialVerify**(vk, m, σ): On input a verification key $vk = (vk_0, \dots, vk_{L-1})$, a message $m \in \mathcal{M}$, and a signature $\sigma = (\sigma_0, \dots, \sigma_{L-1})$, the partial-verification algorithm proceeds as follows:
 1. Output 1 if and only if $\text{PPCTS.PartialVerify}(vk_j, m, \sigma_j) = 1$ for every $j \in \{0, \dots, L-1\}$.
- **Preprocess**($crs, ((vk_1, w_1), \dots, (vk_k, w_k))$): On input a common reference string $crs = (crs_0, \dots, crs_{L-1})$ and a vector of $k \in [2^\lambda]$ verification key and weight pairs (vk_ℓ, w_ℓ) , where $vk_\ell = (vk_{\ell,0}, \dots, vk_{\ell,L-1})$ and $w_\ell \in [0, 2^L - 1]$, the preprocessing algorithm proceeds as follows:
 1. For each $\ell \in [k]$, write the binary decomposition $w_\ell = \sum_{j \in [0, L-1]} 2^j w_{\ell,j}$.
 2. For each $j \in \{0, \dots, L-1\}$:
 - (a) Let $I_j := \{\ell \in [k] : w_{\ell,j} = 1\}$ and $k_j := |I_j|$, and let $\ell_1, \dots, \ell_{k_j}$ be the elements of I_j in ascending order.
 - (b) If $I_j = \emptyset$, set $vk_{\text{agg}}^{(j)} := \perp$ and $ak^{(j)} := \perp$. Otherwise, sample

$$(vk_{\text{agg}}^{(j)}, ak^{(j)}) \leftarrow \text{PPCTS.Preprocess}\left(crs_j, (vk_{\ell_1,j}, \dots, vk_{\ell_{k_j},j})\right).$$
 3. Output $vk_{\text{agg}} := ((vk_{\text{agg}}^{(0)}, k_0), \dots, (vk_{\text{agg}}^{(L-1)}, k_{L-1}))$ and $ak := ((ak^{(0)}, \dots, ak^{(L-1)}), (I_0, \dots, I_{L-1}))$.
- **Aggregate**($ak, m, \{(\ell, \sigma_\ell)\}_{\ell \in T}$): On input the aggregation key $ak = ((ak^{(j)})_{j=0}^{L-1}, (I_j)_{j=0}^{L-1})$, a message $m \in \mathcal{M}$, and a set of index and signature pairs $\{(\ell, \sigma_\ell)\}_{\ell \in T}$ for $T \subseteq [k]$, where $\sigma_\ell = (\sigma_{\ell,0}, \dots, \sigma_{\ell,L-1})$, the aggregation algorithm proceeds as follows:
 1. For each $j \in \{0, \dots, L-1\}$:
 - (a) Write $I_j = \{\ell_1, \dots, \ell_{k_j}\}$ in the same canonical ordering Preprocess used, and set $T_j := \{i \in [k_j] : \ell_i \in T\}$ and $c_j := |T_j| = |T \cap I_j|$. The parties of copy j are indexed by their position in that ordering, which is the index space PPCTS.Aggregate expects.
 - (b) If $c_j > 0$, compute $\sigma_{\text{agg}}^{(j)} = \text{PPCTS.Aggregate}(ak^{(j)}, m, \{(i, \sigma_{\ell_i,j})\}_{i \in T_j})$. Otherwise set $\sigma_{\text{agg}}^{(j)} := \perp$.
 2. Output $\sigma_{\text{agg}} := ((c_0, \sigma_{\text{agg}}^{(0)}), \dots, (c_{L-1}, \sigma_{\text{agg}}^{(L-1)}))$.
- **Verify**($vk_{\text{agg}}, t, m, \sigma_{\text{agg}}$): On input an aggregated verification key $vk_{\text{agg}} = ((vk_{\text{agg}}^{(0)}, k_0), \dots, (vk_{\text{agg}}^{(L-1)}, k_{L-1}))$, a weight threshold $t \in [k \cdot (2^L - 1)]$, a message $m \in \mathcal{M}$, and an aggregate signature $\sigma_{\text{agg}} = ((c_i, \sigma_{\text{agg}}^{(i)})_{i \in [0, L-1]})$, the verification algorithm proceeds as follows:
 1. If $c_j > k_j$ for some j , or if $\sum_{j \in [0, L-1]} 2^j c_j < t$, abort and output 0.
 2. Output 1 if and only if for every j with $c_j > 0$, $\text{PPCTS.Verify}(vk_{\text{agg}}^{(j)}, c_j, m, \sigma_{\text{agg}}^{(j)}) = 1$.
- **Sim**($1^\lambda, 1^L, 1^k, \{w_\ell\}_{\ell \in [k]}, S, m^*$): On input a security parameter λ , a bound on the bit-representation of weights L , a number of parties k , weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a set of corrupted users $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$, the simulator proceeds as follows:
 1. For each $\ell \in [k]$, write the binary decomposition $w_\ell = \sum_{j \in [0, L-1]} 2^j w_{\ell,j}$.
 2. For each $j \in \{0, \dots, L-1\}$:
 - (a) Let $I_j := \{\ell \in [k] : w_{\ell,j} = 1\}$ and $k_j := |I_j|$, and denote $I_j = \{\ell_1, \dots, \ell_{k_j}\}$ for some canonical ordering of the elements of I_j .

- (b) Let $S_j = \{i : \ell_i \in I_j \cap S\}$ be the set of corrupted users in I_j .
(c) If $I_j = \emptyset$, sample $\text{crs}_j \leftarrow \text{PPCTS.Setup}(1^\lambda)$ and set $r_j := \perp$, matching the empty copy that Preprocess skips. Otherwise sample

$$(\text{crs}_j, ((\text{vk}_{\ell_i, j}, \text{sk}_{\ell_i, j}))_{i \in [k_j] \setminus S_j}, r_j) \leftarrow \text{PPCTS.Sim}(1^\lambda, 1^{k_j}, S_j, m^*).$$

- (d) For each $\ell \in [k] \setminus (I_j \cup S)$, sample $(\text{vk}_{\ell, j}, \text{sk}_{\ell, j}) \leftarrow \text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$. These honest parties do not take part in copy j , but still need a key component to answer signing queries, since vk_ℓ and sk_ℓ carry one component per copy.

3. For each $\ell \in [k] \setminus S$, set

$$\text{vk}_\ell := (\text{vk}_{\ell, 0}, \dots, \text{vk}_{\ell, L-1}), \quad \text{sk}_\ell := (\text{sk}_{\ell, 0}, \dots, \text{sk}_{\ell, L-1}).$$

Set $\text{crs} := (\text{crs}_0, \dots, \text{crs}_{L-1})$ and $r := (r_0, \dots, r_{L-1})$, and output

$$(\text{crs}, ((\text{vk}_\ell, \text{sk}_\ell))_{\ell \in [k] \setminus S}, r).$$

We now show that [Construction 8.3](#) is both correct and succinct.

Lemma 8.4 (Correctness). *Suppose that Π_{PPCTS} satisfies signing correctness and aggregation correctness ([Definition 7.1](#)). Then the puncturable public-coin weighted threshold signature scheme Π_{PPCWTs} of [Construction 8.3](#) satisfies signing correctness and aggregation correctness ([Definition 8.1](#)).*

Proof. Throughout this proof, we fix $\lambda, L \in \mathbb{N}$ and $m \in \mathcal{M}$. We have to prove signing correctness and aggregation correctness.

Signing correctness: Fix any $\text{crs} = (\text{crs}_0, \dots, \text{crs}_{L-1})$ in the support of $\text{Setup}(1^\lambda, 1^L)$. Let (vk, sk) be in the support of $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$, so $\text{vk} = (\text{vk}_0, \dots, \text{vk}_{L-1})$ and $\text{sk} = (\text{sk}_0, \dots, \text{sk}_{L-1})$ with each $(\text{vk}_j, \text{sk}_j)$ in the support of $\text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$, and let σ be in the support of $\text{Sign}(\text{sk}, m)$, so $\sigma = (\sigma_0, \dots, \sigma_{L-1})$ with $\sigma_j \leftarrow \text{PPCTS.Sign}(\text{sk}_j, m)$ for each $j \in \{0, \dots, L-1\}$. Signing correctness of Π_{PPCTS} gives $\text{PPCTS.PartialVerify}(\text{vk}_j, m, \sigma_j) = 1$ for every j , so $\text{PartialVerify}(\text{vk}, m, \sigma) = 1$ by construction.

Aggregation correctness: Fix any $k \in [2^\lambda]$, any weights $w_1, \dots, w_k \in [0, 2^L - 1]$, any threshold $t \in [k \cdot (2^L - 1)]$, and any $T \subseteq [k]$ with $\sum_{\ell \in T} w_\ell \geq t$. Let:

- A common reference string $\text{crs} = (\text{crs}_0, \dots, \text{crs}_{L-1})$ in the support of $\text{Setup}(1^\lambda, 1^L)$, so that each crs_j is in the support of $\text{PPCTS.Setup}(1^\lambda)$.
- For each $\ell \in [k]$, a key pair $(\text{vk}_\ell, \text{sk}_\ell) = ((\text{vk}_{\ell, 0}, \dots, \text{vk}_{\ell, L-1}), (\text{sk}_{\ell, 0}, \dots, \text{sk}_{\ell, L-1}))$ in the support of $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$, so that each $(\text{vk}_{\ell, j}, \text{sk}_{\ell, j})$ is in the support of $\text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$.
- For each $\ell \in T$, a signature $\sigma_\ell = (\sigma_{\ell, 0}, \dots, \sigma_{\ell, L-1})$ with $\text{PartialVerify}(\text{vk}_\ell, m, \sigma_\ell) = 1$. By definition of PartialVerify , this means $\text{PPCTS.PartialVerify}(\text{vk}_{\ell, j}, m, \sigma_{\ell, j}) = 1$ for every $j \in \{0, \dots, L-1\}$.
- A verification key $\text{vk}_{\text{agg}} = ((\text{vk}_{\text{agg}}^{(0)}, k_0), \dots, (\text{vk}_{\text{agg}}^{(L-1)}, k_{L-1}))$ and aggregation key $\text{ak} = ((\text{ak}^{(j)})_{j=0}^{L-1}, (I_j)_{j=0}^{L-1})$ in the support of $\text{Preprocess}(\text{crs}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)))$. For each $\ell \in [k]$, we write $w_\ell = \sum_{j \in [0, L-1]} 2^j w_{\ell, j}$ and set $I_j := \{\ell \in [k] : w_{\ell, j} = 1\}$. Denote $k_j := |I_j|$ and $I_j = \{\ell_1, \dots, \ell_{k_j}\}$ for some canonical ordering. Then $(\text{vk}_{\text{agg}}^{(j)}, \text{ak}^{(j)}) = \text{PPCTS.Preprocess}(\text{crs}_j, (\text{vk}_{\ell_1, j}, \dots, \text{vk}_{\ell_{k_j}, j}); r_j)$ for each j with $I_j \neq \emptyset$, and $\text{vk}_{\text{agg}}^{(j)} = \text{ak}^{(j)} = \perp$ for each j with $I_j = \emptyset$.
- An aggregate signature $\sigma_{\text{agg}} = ((c_0, \sigma_{\text{agg}}^{(0)}), \dots, (c_{L-1}, \sigma_{\text{agg}}^{(L-1)})) = \text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T})$, where $T_j := \{i \in [k_j] : \ell_i \in T\}$ and $c_j := |T_j| = |T \cap I_j|$, and where $\sigma_{\text{agg}}^{(j)} = \text{PPCTS.Aggregate}(\text{ak}^{(j)}, m, \{(i, \sigma_{\ell_i, j})\}_{i \in T_j})$ for each j with $c_j > 0$ and $\sigma_{\text{agg}}^{(j)} = \perp$ for each j with $c_j = 0$.

First, $\sum_{j \in [0, L-1]} 2^j c_j \geq t$: since $I_j = \{\ell \in [k] : w_{\ell, j} = 1\}$ and each $w_{\ell, j} \in \{0, 1\}$, we have $c_j = |T \cap I_j| = \sum_{\ell \in T} w_{\ell, j}$, so

$$\sum_{j \in [0, L-1]} 2^j c_j = \sum_{j \in [0, L-1]} 2^j \sum_{\ell \in T} w_{\ell, j} = \sum_{\ell \in T} \sum_{j \in [0, L-1]} 2^j w_{\ell, j} = \sum_{\ell \in T} w_{\ell} \geq t.$$

And $c_j = |T \cap I_j| \leq |I_j| = k_j$ for every j , so Verify does not abort on the checks in [Step 1](#).

Second, we want to argue that the final check in Verify passes. This means that for any j with $c_j > 0$, we want to prove that $\text{PPCTS.Verify}(\text{vk}_{\text{agg}}^{(j)}, c_j, m, \sigma_{\text{agg}}^{(j)}) = 1$. Since $c_j > 0$ and $c_j = k_j$, then $I_j \neq \emptyset$. For every $i \in T_j$ the party ℓ_i lies in I_j , so $\text{vk}_{\ell_i, j}$ is the i^{th} entry of the vector fed into PPCTS.Preprocess , and lies in T , so $\text{PPCTS.PartialVerify}(\text{vk}_{\ell_i, j}, m, \sigma_{\ell_i, j}) = 1$. Thus $\{\sigma_{\ell_i, j}\}_{i \in T_j}$ are valid partial signatures under the very keys that produced $(\text{vk}_{\text{agg}}^{(j)}, \text{ak}^{(j)})$, the number of keys satisfies $k_j \in [2^\lambda]$, as $0 < c_j \leq k_j \leq k \leq 2^\lambda$, the selected set $T_j \subseteq [k_j]$ has size c_j , and the threshold satisfies $c_j \in [k_j]$. Aggregation correctness of Π_{PPCTS} , applied to the k_j keys of copy j with threshold c_j and set T_j , gives $\text{PPCTS.Verify}(\text{vk}_{\text{agg}}^{(j)}, c_j, m, \sigma_{\text{agg}}^{(j)}) = 1$.

Combining the two, $\text{Verify}(\text{vk}_{\text{agg}}, t, m, \sigma_{\text{agg}}) = 1$. $\square$

Lemma 8.5 (Succinctness). *Suppose that Π_{PPCTS} is succinct ([Definition 7.1](#)), witnessed by a polynomial $\text{poly}_{\text{PPCTS}}(\cdot, \cdot)$. Then the puncturable public-coin weighted threshold signature scheme Π_{PPCWTs} of [Construction 8.3](#) is succinct.*

Proof. Throughout this proof, we fix $\lambda, L \in \mathbb{N}, k \in [2^\lambda]$, a threshold $t \in [k \cdot (2^L - 1)]$, and weights $w_1, \dots, w_k \in [0, 2^L - 1]$. We take the polynomial $\text{poly}_{\text{PPCTS}}$ that witnesses succinctness of Π_{PPCTS} to be monotone in both of its arguments and at least the constant $\lfloor \lambda \rfloor$, as we may. Let:

- A common reference string $\text{crs} = (\text{crs}_0, \dots, \text{crs}_{L-1})$ in the support of $\text{Setup}(1^\lambda, 1^L)$, so that each crs_j is in the support of $\text{PPCTS.Setup}(1^\lambda)$.
- For each $\ell \in [k]$, a key pair $(\text{vk}_\ell, \text{sk}_\ell) = ((\text{vk}_{\ell, 0}, \dots, \text{vk}_{\ell, L-1}), (\text{sk}_{\ell, 0}, \dots, \text{sk}_{\ell, L-1}))$ in the support of $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$, so that each $(\text{vk}_{\ell, j}, \text{sk}_{\ell, j})$ is in the support of $\text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$.
- A verification key $\text{vk}_{\text{agg}} = ((\text{vk}_{\text{agg}}^{(0)}, k_0), \dots, (\text{vk}_{\text{agg}}^{(L-1)}, k_{L-1}))$ and aggregation key $\text{ak} = ((\text{ak}^{(j)})_{j=0}^{L-1}, (I_j)_{j=0}^{L-1})$ in the support of $\text{Preprocess}(\text{crs}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)))$. For each $\ell \in [k]$, we write $w_\ell = \sum_{j \in [0, L-1]} 2^j w_{\ell, j}$ and set $I_j := \{\ell \in [k] : w_{\ell, j} = 1\}$. Denote $k_j := |I_j|$ and $I_j = \{\ell_1, \dots, \ell_{k_j}\}$ for some canonical ordering. Then $(\text{vk}_{\text{agg}}^{(j)}, \text{ak}^{(j)})$ is in the support of $\text{PPCTS.Preprocess}(\text{crs}_j, (\text{vk}_{\ell_1, j}, \dots, \text{vk}_{\ell_{k_j}, j}))$ for each j with $I_j \neq \emptyset$, and $\text{vk}_{\text{agg}}^{(j)} = \text{ak}^{(j)} = \perp$ for each j with $I_j = \emptyset$.
- A message $m \in \mathcal{M}$, a set $T \subseteq [k]$, signatures $\sigma_\ell = (\sigma_{\ell, 0}, \dots, \sigma_{\ell, L-1})$ for $\ell \in T$, and the aggregate signature $\sigma_{\text{agg}} = ((c_0, \sigma_{\text{agg}}^{(0)}), \dots, (c_{L-1}, \sigma_{\text{agg}}^{(L-1)})) = \text{Aggregate}(\text{ak}, m, \{(\ell, \sigma_\ell)\}_{\ell \in T})$, where $T_j := \{i \in [k_j] : \ell_i \in T\}$ and $c_j := |T_j| = |T \cap I_j|$, and where $\sigma_{\text{agg}}^{(j)} = \text{PPCTS.Aggregate}(\text{ak}^{(j)}, m, \{(i, \sigma_{\ell_i, j})\}_{i \in T_j})$ for each j with $c_j > 0$ and $\sigma_{\text{agg}}^{(j)} = \perp$ for each j with $c_j = 0$.

First, $|\text{vk}_{\text{agg}}| \leq \text{poly}(\lambda, \log k, L)$. Every copy has $k_j = |I_j| \leq k \leq 2^\lambda$, so for each j with $I_j \neq \emptyset$, succinctness of Π_{PPCTS} applies to the k_j keys of copy j and gives $|\text{vk}_{\text{agg}}^{(j)}| \leq \text{poly}_{\text{PPCTS}}(\lambda, \log k_j) \leq \text{poly}_{\text{PPCTS}}(\lambda, \log k)$; and for each j with $I_j = \emptyset$, $|\text{vk}_{\text{agg}}^{(j)}| = |\perp| \leq \text{poly}_{\text{PPCTS}}(\lambda, \log k)$. Since the size $k_j \leq k$ of each copy is representable in $\lceil \log(k+1) \rceil$ bits,

$$|\text{vk}_{\text{agg}}| = \sum_{j \in [0, L-1]} (|\text{vk}_{\text{agg}}^{(j)}| + |k_j|) \leq L \cdot (\text{poly}_{\text{PPCTS}}(\lambda, \log k) + \lceil \log(k+1) \rceil).$$

Second, $|\sigma_{\text{agg}}| \leq \text{poly}(\lambda, \log k, L)$. Fix any j . The count $c_j = |T \cap I_j| \leq k_j \leq k$ is again representable in $\lceil \log(k+1) \rceil$ bits. If $c_j > 0$, then $I_j \neq \emptyset$ and $\sigma_{\text{agg}}^{(j)}$ is an aggregate signature output by $\text{PPCTS.Aggregate}(\text{ak}^{(j)}, \cdot, \cdot)$ for the

$(\text{vk}_{\text{agg}}^{(j)}, \text{ak}^{(j)})$ above, so succinctness of Π_{PPCTS} gives $|\sigma_{\text{agg}}^{(j)}| \leq \text{poly}_{\text{PPCTS}}(\lambda, \log k_j) \leq \text{poly}_{\text{PPCTS}}(\lambda, \log k)$; and if $c_j = 0$, then $|\sigma_{\text{agg}}^{(j)}| = |\perp| \leq \text{poly}_{\text{PPCTS}}(\lambda, \log k)$. Hence

$$|\sigma_{\text{agg}}| = \sum_{j \in [0, L-1]} (|c_j| + |\sigma_{\text{agg}}^{(j)}|) \leq L \cdot (\text{poly}_{\text{PPCTS}}(\lambda, \log k) + \lceil \log(k+1) \rceil).$$

Combining the two, both sizes are bounded by $\text{poly}(\lambda, \log k, L) := L \cdot (\text{poly}_{\text{PPCTS}}(\lambda, \log k) + \log k + 1)$, a single polynomial that is universal because $\text{poly}_{\text{PPCTS}}$ is, so Π_{PPCWTs} is succinct. $\square$

Theorem 8.6 (Punctured Security). *Suppose that Π_{PPCTS} satisfies punctured static security (Definition 7.1). Then the puncturable public-coin weighted threshold signature scheme Π_{PPCWTs} of Construction 8.3, with Sim as defined there, satisfies punctured static security (Definition 8.1).*

Proof. Let $\mathcal{A}$ be an (unbounded) adversary for the punctured static security game of Definition 8.1 and $L := L(\lambda)$ be a polynomial. We assume that $\mathcal{A}$ submits admissible weights $\{w_\ell\}_{\ell \in [k]}$ such that there exists $w_\ell > 0$ (otherwise all verification keys produced by algorithm Preprocess are $\perp$, and we assume those are immediately rejected by the underlying scheme Π_{PPCTS}). We construct an adversary $\mathcal{B}$ for the punctured static security game of Π_{PPCTS} as follows:

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ to get a number of parties 1^k , weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a set of corrupted users $S \subseteq [k]$ and a message $m^* \in \mathcal{M}$.
2. For each $j \in \{0, \dots, L-1\}$, algorithm $\mathcal{B}$ lets $I_j := \{\ell \in [k] : w_{\ell,j} = 1\}$ and $k_j := |I_j|$, denotes $I_j = \{\ell_1, \dots, \ell_{k_j}\}$ using the same canonical ordering as Construction 8.3, and lets $S_j := \{i \in [k_j] : \ell_i \in S\}$ be the positions of the corrupted parties within that ordering. It then guesses a copy $j^* \xleftarrow{R} \mathcal{J}$, where $\mathcal{J} := \{j : I_j \neq \emptyset\}$ is the set of nonempty copies, which are the only ones Construction 8.3 preprocesses. Recall that we assume that $\mathcal{J} \neq \emptyset$, so the guess is well-defined.
3. Algorithm $\mathcal{B}$ sends $1^{k_{j^*}}, S_{j^*}$, and m^* to its own challenger and receives $\text{crs}_{j^*}, (\text{vk}_{\ell_i, j^*})_{i \in [k_{j^*}] \setminus S_{j^*}}$, and r_{j^*} .
4. Algorithm $\mathcal{B}$ simulates the common reference string and the keys. For each $j \in \{0, \dots, L-1\} \setminus \{j^*\}$ with $I_j \neq \emptyset$, it samples

$$(\text{crs}_j, ((\text{vk}_{\ell_i, j}, \text{sk}_{\ell_i, j}))_{i \in [k_j] \setminus S_j}, r_j) \leftarrow \text{PPCTS.Sim}(1^\lambda, 1^{k_j}, S_j, m^*),$$
 and for each empty copy it samples $\text{crs}_j \leftarrow \text{PPCTS.Setup}(1^\lambda)$ and sets $r_j := \perp$. For every j and every $\ell \in [k] \setminus (I_j \cup S)$ it samples $(\text{vk}_{\ell, j}, \text{sk}_{\ell, j}) \leftarrow \text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$.
5. Algorithm $\mathcal{B}$ sets $\text{crs} := (\text{crs}_0, \dots, \text{crs}_{L-1})$, $\text{vk}_\ell := (\text{vk}_{\ell, 0}, \dots, \text{vk}_{\ell, L-1})$ for each $\ell \in [k] \setminus S$, and $r := (r_0, \dots, r_{L-1})$, and sends $\text{crs}, (\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$. Every honest verification-key component is available from the challenger or from step 4; the missing honest signing-key components are handled by its signing oracle.
6. Whenever $\mathcal{A}$ makes a signing query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, algorithm $\mathcal{B}$ computes $\sigma_{\ell, j} \leftarrow \text{PPCTS.Sign}(\text{sk}_{\ell, j}, m)$ itself for every $j \neq j^*$. For $j = j^*$: if $\ell \notin I_{j^*}$ it likewise signs with the key sk_{ℓ, j^*} it sampled in step 4; and if $\ell = \ell_i \in I_{j^*}$, so that $i \in [k_{j^*}] \setminus S_{j^*}$, it forwards (i, m) to its own signing oracle to obtain σ_{ℓ, j^*} . It replies with $\sigma_\ell := (\sigma_{\ell, 0}, \dots, \sigma_{\ell, L-1})$.
7. Algorithm $\mathcal{A}$ outputs arbitrary corrupted verification keys $(\text{vk}_\ell)_{\ell \in S}$ (not necessarily generated by KeyGen), a threshold $t \in [k \cdot (2^L - 1)]$ with $t > \sum_{\ell \in S} w_\ell$, and a forgery $\sigma_{\text{agg}} = ((c_j, \sigma_{\text{agg}}^{(j)}))_{j=0}^{L-1}$. If $c_{j^*} \leq |S_{j^*}|$ or $c_{j^*} > k_{j^*}$, algorithm $\mathcal{B}$ aborts. Otherwise it parses each corrupted key $\text{vk}_\ell = (\text{vk}_{\ell, 0}, \dots, \text{vk}_{\ell, L-1})$ and outputs $(\text{vk}_{\ell_i, j^*})_{i \in S_{j^*}}$, the threshold c_{j^*} , and the forgery $\sigma_{\text{agg}}^{(j^*)}$.

The simulation is perfect and independent of j^* . For every copy $j \neq j^*$, algorithm $\mathcal{B}$ samples crs_j , the honest key pairs of $I_j \setminus S$ and r_j through $\text{PPCTS.Sim}(1^\lambda, 1^{k_j}, S_j, m^*)$ when $I_j \neq \emptyset$ and through $\text{PPCTS.Setup}(1^\lambda)$ with $r_j = \perp$ when $I_j = \emptyset$, and the keys of $[k] \setminus (I_j \cup S)$ through PPCTS.KeyGen , exactly as Sim of Construction 8.3 does.

For copy j^* , the challenger uses PPCTS.Sim with the same parameters to generate crs_{j^*} , the honest key pairs, and r_{j^*} . It returns crs_{j^*} , the honest verification keys, and r_{j^*} , while $\mathcal{B}$ samples the keys of $[k] \setminus (I_{j^*} \cup S)$ itself. Signing queries are answered with those very keys in both cases, and the ones $\mathcal{B}$ forwards are legal for its own oracle: they carry an index $i \in [k_{j^*}] \setminus S_{j^*}$ and a message $m \neq m^*$, as [Definition 8.1](#) restricts $\mathcal{A}$ to $m \neq m^*$. So $\mathcal{A}$'s view is distributed exactly as in the punctured static security game of [Definition 8.1](#) and is the same for every value of j^* . In particular j^* is uniform on $\mathcal{J}$ conditioned on $\mathcal{A}$'s output.

The reduction breaks Π_{PPCTS} . Suppose $\mathcal{A}$ wins, that is, $\text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}}) = 1$ for the aggregated key $(\text{vk}_{\text{agg}}, \text{ak}) = \text{Preprocess}(\text{crs}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)); r)$ that the challenger computes. The first check of Verify gives $c_j \leq k_j$ for every j and $\sum_j 2^j c_j \geq t$, and the second gives $\text{PPCTS.Verify}(\text{vk}_{\text{agg}}^{(j)}, c_j, m^*, \sigma_{\text{agg}}^{(j)}) = 1$ for every j with $c_j > 0$. We claim there is a copy $\hat{j}$ with $c_{\hat{j}} > |S_{\hat{j}}|$. Otherwise $c_j \leq |S_j| = \sum_{\ell \in S} w_{\ell, j}$ for every j , and then

$$t \leq \sum_{j \in [0, L-1]} 2^j c_j \leq \sum_{j \in [0, L-1]} 2^j \sum_{\ell \in S} w_{\ell, j} = \sum_{\ell \in S} \sum_{j \in [0, L-1]} 2^j w_{\ell, j} = \sum_{\ell \in S} w_{\ell} < t,$$

using $t > \sum_{\ell \in S} w_{\ell}$ in the last step. Fix such a $\hat{j}$. Then $c_{\hat{j}} > |S_{\hat{j}}| \geq 0$, so Verify did check that copy and therefore $\text{PPCTS.Verify}(\text{vk}_{\text{agg}}^{(\hat{j})}, c_{\hat{j}}, m^*, \sigma_{\text{agg}}^{(\hat{j})}) = 1$, implying $k_{\hat{j}} \geq c_{\hat{j}} > 0$, so $\hat{j} \in \mathcal{J}$ and $c_{\hat{j}} \in [k_{\hat{j}}]$.

Suppose that $j^* = \hat{j}$. By the choice of $\hat{j}$ and the fact that $\mathcal{A}$ wins, neither abort condition of step 7 is triggered, so $\mathcal{B}$ outputs the threshold $c_{j^*} \in [k_{j^*}]$ with $c_{j^*} > |S_{j^*}|$, which is a legal output in the game of [Definition 7.1](#), together with $\sigma_{\text{agg}}^{(j^*)}$ and the corrupted key components $(\text{vk}_{\ell, j^*})_{i \in S_{j^*}}$ supplied by $\mathcal{A}$. Its own challenger scores that forgery against

$$(\text{vk}'_{\text{agg}}, \text{ak}') = \text{PPCTS.Preprocess}(\text{crs}_{j^*}, (\text{vk}_{\ell_1, j^*}, \dots, \text{vk}_{\ell_{k_{j^*}}, j^*}); r_{j^*}).$$

The weighted preprocessing algorithm computes $\text{vk}_{\text{agg}}^{(j^*)}$ using this same call: it uses the same crs_{j^*} , the same key vector (including the corrupted components supplied by $\mathcal{A}$) in the same canonical ordering, and the same randomness r_{j^*} . Hence $\text{vk}'_{\text{agg}} = \text{vk}_{\text{agg}}^{(j^*)}$ and the challenger's output is $\text{PPCTS.Verify}(\text{vk}_{\text{agg}}^{(j^*)}, c_{j^*}, m^*, \sigma_{\text{agg}}^{(j^*)}) = 1$.

By the perfect simulation, the index j^* is uniform on $\mathcal{J}$ and independent of the rest of the experiment, and $|\mathcal{J}| \leq L$, so

$$\begin{aligned} \Pr[\mathcal{B} \text{ wins}] &\geq \Pr[j^* = \hat{j} \wedge \text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}}) = 1] \\ &= \frac{1}{|\mathcal{J}|} \cdot \Pr[\text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}}) = 1] \\ &\geq \frac{1}{L} \cdot \Pr[\text{Verify}(\text{vk}_{\text{agg}}, t, m^*, \sigma_{\text{agg}}) = 1]. \end{aligned}$$

Since L is polynomial, if $\mathcal{A}$ wins with non-negligible probability then so does $\mathcal{B}$ and the claim follows by the punctured static security of Π_{PPCTS} . $\square$

Theorem 8.7 (Mode Indistinguishability). *Suppose that Π_{PPCTS} satisfies mode indistinguishability ([Definition 7.1](#)). Then the puncturable public-coin weighted threshold signature scheme Π_{PPCWTs} of [Construction 8.3](#), with Sim as defined there, satisfies mode indistinguishability ([Definition 8.1](#)).*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game of [Definition 8.1](#). Note that this game never invokes Preprocess : the challenger sends $\mathcal{A}$ only crs , the honest parties' verification keys and the randomness r , and answers signing queries, so no aggregated key is ever formed. Since $\mathcal{A}$ is efficient and writes the weight bound in unary, there is a polynomial $L_{\max} = L_{\max}(\lambda)$, depending only on $\mathcal{A}$, such that every first message of $\mathcal{A}$ satisfies $L \leq L_{\max}$. Throughout, once $\mathcal{A}$ has sent its first message we write, for each copy $j \in \{0, \dots, L-1\}$, $I_j := \{\ell \in [k] : w_{\ell, j} = 1\} = \{\ell_1, \dots, \ell_{k_j}\}$ in the canonical ordering of [Construction 8.3](#), $k_j := |I_j|$, and $S_j := \{i \in [k_j] : \ell_i \in S\}$, exactly as Sim does.

We define the following experiments; all of them output $\mathcal{A}$'s guess b' .

- Real_0 : the mode indistinguishability game of [Definition 8.1](#) with $b = 0$. In full:

1. Algorithm $\mathcal{A}(1^\lambda)$ outputs $1^L, 1^k$, weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a set $S \subseteq [k]$, and a message $m^* \in \mathcal{M}$.
 2. For each $j \in \{0, \dots, L-1\}$, the challenger samples $\text{crs}_j \leftarrow \text{PPCTS.Setup}(1^\lambda)$, samples $(\text{vk}_{\ell,j}, \text{sk}_{\ell,j}) \leftarrow \text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$ for every $\ell \in [k] \setminus S$, and samples r_j uniformly at random, as preprocessing coins for k_j participants under crs_j ; their length does not depend on the verification-key values. For an empty copy, preprocessing is skipped and $r_j = \perp$.
 3. The challenger sets $\text{crs} := (\text{crs}_0, \dots, \text{crs}_{L-1})$, $\text{vk}_\ell := (\text{vk}_{\ell,0}, \dots, \text{vk}_{\ell,L-1})$ and $\text{sk}_\ell := (\text{sk}_{\ell,0}, \dots, \text{sk}_{\ell,L-1})$ for each $\ell \in [k] \setminus S$, and $r := (r_0, \dots, r_{L-1})$, and sends $\text{crs}, (\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$.
 4. Throughout the remainder of the experiment, $\mathcal{A}$ may adaptively make signing queries: on a query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$, the challenger replies with $\text{Sign}(\text{sk}_\ell, m)$.
 5. Algorithm $\mathcal{A}$ outputs b' , the output of the experiment.
- Real_1 : the mode indistinguishability game of [Definition 8.1](#) with $b = 1$. Same as Real_0 , except that step 2 becomes:
 2. For each nonempty copy j , the challenger samples

$$(\text{crs}_j, ((\text{vk}_{\ell,j}, \text{sk}_{\ell,j}))_{\ell \in [k_j] \setminus S_j}, r_j) \leftarrow \text{PPCTS.Sim}(1^\lambda, 1^{k_j}, S_j, m^*).$$

For an empty copy, it samples $\text{crs}_j \leftarrow \text{PPCTS.Setup}(1^\lambda)$ and sets $r_j := \perp$. For every copy j , it samples $(\text{vk}_{\ell,j}, \text{sk}_{\ell,j}) \leftarrow \text{PPCTS.KeyGen}(1^\lambda, \text{crs}_j)$ for every $\ell \in [k] \setminus (I_j \cup S)$.

Comparing with [Construction 8.3](#) copy by copy, Real_0 samples crs and the keys exactly as $\text{Setup}(1^\lambda, 1^L)$ and $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$ do and r uniformly, so it is the game with $b = 0$; and Real_1 samples crs , the keys and r exactly as $\text{Sim}(1^\lambda, 1^L, 1^k, \{w_\ell\}_{\ell \in [k]}, S, m^*)$ does, so it is the game with $b = 1$. The two differ only in how, for each nonempty copy j , the triple consisting of crs_j , the keys of the honest participants $I_j \setminus S$, and r_j is drawn: from PPCTS.Setup , PPCTS.KeyGen and uniform coins in Real_0 , and from $\text{PPCTS.Sim}(1^\lambda, 1^{k_j}, S_j, m^*)$ in Real_1 . These are precisely the two modes of [Definition 7.1](#) for a scheme with k_j parties and corrupted set S_j . On an empty copy the two agree, since then both sample $\text{crs}_j \leftarrow \text{PPCTS.Setup}(1^\lambda)$, set $r_j = \perp$, and generate a key for every honest party using PPCTS.KeyGen .

For $p \in \{0, \dots, L\}$, define Hyb_p to be the experiment that generates copy j as Real_1 does for every $j < p$ and as Real_0 does for every $j \geq p$, and is otherwise unchanged. Then $\text{Hyb}_0 = \text{Real}_0$, $\text{Hyb}_L = \text{Real}_1$, and $\text{Hyb}_p = \text{Hyb}_{p+1}$ whenever $I_p = \emptyset$.

Claim 8.8. Assume Π_{PPCTS} satisfies mode indistinguishability. Then for any efficient $\mathcal{A}$ there is a negligible negl such that

$$|\Pr[\text{Real}_0(\mathcal{A}) = 1] - \Pr[\text{Real}_1(\mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

Proof. For $p \in \{L, \dots, L_{\max}\}$ we additionally let $\text{Hyb}_p := \text{Hyb}_L$, so that Hyb_p is defined for every $p \in \{0, \dots, L_{\max}\}$. Consider the following adversary $\mathcal{B}$ for the mode indistinguishability game of [Definition 7.1](#).

1. Sample $p^* \xleftarrow{\mathcal{R}} \{0, \dots, L_{\max} - 1\}$.
2. Run $\mathcal{A}(1^\lambda)$ to obtain $1^L, 1^k, \{w_\ell\}_{\ell \in [k]}, S, m^*$, and compute I_j, k_j and S_j for every copy j .
3. If $p^* \geq L$ or $I_{p^*} = \emptyset$, send the dummy first message $1^1, \emptyset, m^*$ to the challenger, discard the answer, complete the experiment $\text{Hyb}_{\min(p^*, L)}$ on its own, and output $\mathcal{A}$'s guess. Otherwise continue.
4. Send $1^{k_{p^*}}, S_{p^*}$, and m^* to the challenger. The challenger returns crs_{p^*} and r_{p^*} together with the honest verification keys $(\text{vk}_{\ell, p^*})_{\ell \in [k_{p^*}] \setminus S_{p^*}}$. This is admissible, as $1 \leq k_{p^*} \leq k \leq 2^\lambda$ and $S_{p^*} \subseteq [k_{p^*}]$.
5. Sample $(\text{vk}_{\ell, p^*}, \text{sk}_{\ell, p^*}) \leftarrow \text{PPCTS.KeyGen}(1^\lambda, \text{crs}_{p^*})$ for every $\ell \in [k] \setminus (I_{p^*} \cup S)$, and generate every copy $j \neq p^*$ on its own: as in step 2 of Real_1 if $j < p^*$, and as in step 2 of Real_0 if $j > p^*$.
6. Set $\text{crs} := (\text{crs}_0, \dots, \text{crs}_{L-1})$, $\text{vk}_\ell := (\text{vk}_{\ell,0}, \dots, \text{vk}_{\ell,L-1})$ for each $\ell \in [k] \setminus S$, and $r := (r_0, \dots, r_{L-1})$, and send $\text{crs}, (\text{vk}_\ell)_{\ell \in [k] \setminus S}$ and r to $\mathcal{A}$. Every honest verification-key component is available; signing queries for honest participants in copy p^* are handled by the challenge signing oracle.

7. On a signing query (ℓ, m) with $\ell \in [k] \setminus S$ and $m \neq m^*$: for every copy $j \neq p^*$ compute $\sigma_{\ell,j} \leftarrow \text{PPCTS.SignKey}(\text{sk}_{\ell,j}, m)$ with the sampled keys; for copy p^* , do the same with the key from step 5 if $\ell \notin I_{p^*}$, and if $\ell = \ell_i \in I_{p^*}$, so that $i \in [k_{p^*}] \setminus S_{p^*}$, forward (i, m) to the challenge signing oracle instead. Reply with $\sigma_\ell := (\sigma_{\ell,0}, \dots, \sigma_{\ell,L-1})$.
8. Output $\mathcal{A}$'s guess.

Algorithm $\mathcal{B}$ is efficient: it runs $\mathcal{A}$ once and samples at most L copies' worth of keys, and the queries it forwards are legal, carrying an index $i \in [k_{p^*}] \setminus S_{p^*}$ and, since [Definition 8.1](#) restricts $\mathcal{A}$ to $m \neq m^*$, a message $m \neq m^*$.

Fix a value $p^* \in \{0, \dots, L_{\max} - 1\}$ of the sampled index. On the runs that take step 3, algorithm $\mathcal{B}$ simulates $\text{Hyb}_{\min(p^*, L)}$ on its own, regardless of the challenger's bit; and there $\text{Hyb}_{p^*} = \text{Hyb}_{p^*+1}$, either because both equal Hyb_L when $p^* \geq L$, or because $I_{p^*} = \emptyset$. On the other runs, $\mathcal{B}$ generates the copies $j \neq p^*$ as Hyb_{p^*} and Hyb_{p^*+1} both do, and copy p^* from its challenger: in mode $b_{\text{PPCTS}} = 0$ that copy is real, which is exactly Hyb_{p^*} , and in mode $b_{\text{PPCTS}} = 1$ it is simulated, which is exactly Hyb_{p^*+1} . So in both cases, conditioned on the sampled index being p^* , algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\text{Hyb}_{p^*}(\mathcal{A}) = 1]$ when the challenger's bit is 0 and with probability $\Pr[\text{Hyb}_{p^*+1}(\mathcal{A}) = 1]$ when it is 1 (using $\text{Hyb}_{L_{\max}} = \text{Hyb}_L$ to make sense of $p^* + 1 = L_{\max}$).

We bound the signed sum over the inner hybrids rather than each term separately, exactly as in the analogous argument of [Claim 7.10](#): applying the guarantee to each p^* would yield $L_{\max}$ negligible functions, and $L_{\max}$ grows with λ , so their maximum need not be negligible. Since p^* is sampled uniformly from $\{0, \dots, L_{\max} - 1\}$ and independently of the challenger's bit, mode indistinguishability of Π_{PPCTS} gives a single negligible negl' with

$$\begin{aligned} \text{negl}'(\lambda) &\geq \left| \frac{1}{L_{\max}} \sum_{p \in [0, L_{\max}-1]} \left(\Pr[\text{Hyb}_p(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{p+1}(\mathcal{A}) = 1] \right) \right| \\ &= \frac{1}{L_{\max}} |\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{L_{\max}}(\mathcal{A}) = 1]|, \end{aligned}$$

where the sum telescopes to its first and last terms, and $\text{Hyb}_{L_{\max}} = \text{Hyb}_L$. Since $\text{Hyb}_0 = \text{Real}_0$ and $\text{Hyb}_L = \text{Real}_1$, the claim follows with $\text{negl} = L_{\max} \cdot \text{negl}'$, which is negligible because $L_{\max} \leq \text{poly}(\lambda)$. $\square$

Since Real_0 is exactly the mode indistinguishability game of [Definition 8.1](#) with $b = 0$ and Real_1 is exactly the game with $b = 1$, [Claim 8.8](#) establishes mode indistinguishability. $\square$

9 Public-Coin Weighted Threshold Encryption

In this section, we construct a weighted threshold encryption scheme. Similar to before, we work in the distributed setting with a public-coin preprocessing algorithm.

The setup algorithm outputs only a common reference string, and every party samples its own key pair, without knowing how many other parties there are, or what weight it will be given. A public preprocessing algorithm then assigns the parties their weights and compresses any list of public key and weight pairs into a single encryption key, together with an aggregation key that lets anyone combine the parties' decryption hints, and encryption is performed with respect to that encryption key and a weight threshold.

Definition 9.1 (Public-Coin Weighted Threshold Encryption). A public-coin weighted threshold encryption scheme Π_{PCTE} for a message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\text{PCTE}} = (\text{Setup}, \text{KeyGen}, \text{Preprocess}, \text{Enc}, \text{GetHint}, \text{Dec})$ with the following syntax:

- $\text{Setup}(1^\lambda, 1^L) \rightarrow \text{crs}$: On input a security parameter λ and a bound on the bit-representation of weights L , the setup algorithm outputs a common reference string crs . The number of parties is not an input, so a single crs serves every number of parties.
- $\text{KeyGen}(1^\lambda, 1^L, \text{crs}) \rightarrow (\text{sk}, \text{pk})$: On input a security parameter λ , a bound on the bit-representation of weights L , and a common reference string crs , the key-generation algorithm outputs a secret key sk and a public key pk . Each party runs this on its own, independently of the others, and in particular before its weight is known.

- $\text{Preprocess}(\text{crs}, ((\text{pk}_1, w_1), \dots, (\text{pk}_k, w_k))) \rightarrow (\text{ek}, \text{ak})$: On input a common reference string crs and a vector of $k \in [2^\lambda]$ public key and weight pairs (pk_i, w_i) such that $w_i \in [0, 2^L - 1]$, the preprocessing algorithm assigns the parties their weights and outputs an encryption key ek and an aggregation key ak . The length of its random tape is determined by crs and the weights.
- $\text{Enc}(\text{ek}, t, \text{msg}) \rightarrow \text{ct}$: On input an encryption key ek , a weight threshold $t \in [k \cdot (2^L - 1)]$, and a message $\text{msg} \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct .
- $\text{GetHint}(\text{crs}, \text{sk}_i, \text{ct}) \rightarrow \text{dh}_i$: On input a common reference string crs , a secret key sk_i , and a ciphertext ct , the hint algorithm outputs a decryption hint dh_i .
- $\text{Dec}(\text{ak}, \{(i, \text{dh}_i)\}_{i \in T}, \text{ct}) \rightarrow \text{msg}$: On input an aggregation key ak , a set of indices together with the decryption hint of each party in $T \subseteq [k]$, and a ciphertext ct , the decryption algorithm outputs a message msg .

Moreover, Π_{PCTE} should satisfy the following properties:

- **Correctness:** For any $\lambda, L \in \mathbb{N}$, any $k \in [2^\lambda]$, any $\text{msg} \in \mathcal{M}$, any weights $w_1, \dots, w_k \in [0, 2^L - 1]$, any threshold $t \in [k \cdot (2^L - 1)]$, and any subset $T \subseteq [k]$ with $\sum_{i \in T} w_i \geq t$, the following holds:

$$\Pr \left[\begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^L) \\ (\text{sk}_i, \text{pk}_i) \leftarrow \text{KeyGen}(1^\lambda, 1^L, \text{crs}) \text{ for all } i \in [k] \\ (\text{ek}, \text{ak}) \leftarrow \text{Preprocess}(\text{crs}, ((\text{pk}_1, w_1), \dots, (\text{pk}_k, w_k))) \\ \text{ct} \leftarrow \text{Enc}(\text{ek}, t, \text{msg}) \\ \text{dh}_i \leftarrow \text{GetHint}(\text{crs}, \text{sk}_i, \text{ct}) \text{ for all } i \in T \end{array} \right] = 1$$

- **Public-coin static security:** For any adversary $\mathcal{A}$, we define the following experiment parameterized by the security parameter λ and a bit $b \in \{0, 1\}$:

1. On input a security parameter 1^λ , the adversary sends a bound on the bit-representation of weights 1^L , a number of parties 1^k , weights $w_1, \dots, w_k \in [0, 2^L - 1]$, and a set $S \subseteq [k]$ to the challenger.
2. The challenger samples the following:
 - A common reference string $\text{crs} \leftarrow \text{Setup}(1^\lambda, 1^L)$.
 - For every $i \in [k] \setminus S$: a key pair $(\text{sk}_i, \text{pk}_i) \leftarrow \text{KeyGen}(1^\lambda, 1^L, \text{crs})$.
 - An independent random string $r \xleftarrow{\mathcal{R}} \{0, 1\}^{\ell_r}$, where ℓ_r is the prescribed randomness length for Preprocess on crs and these k weights, independent of the public-key values.

The challenger sends crs , r , and $(\text{pk}_i)_{i \in [k] \setminus S}$ to $\mathcal{A}$.

3. The adversary can make hint queries of the form (i, ct) with $i \in [k] \setminus S$. The challenger responds with $\text{dh} \leftarrow \text{GetHint}(\text{crs}, \text{sk}_i, \text{ct})$.
4. The adversary sends arbitrary public keys $(\text{pk}_i)_{i \in S}$ for the corrupted parties (not necessarily generated by KeyGen), a weight threshold $t \in [k \cdot (2^L - 1)]$ with $t > \sum_{i \in S} w_i$ and a pair of messages $\text{msg}_0, \text{msg}_1 \in \mathcal{M}$ of the same length to the challenger. The challenger combines these keys with the honest public keys and, using the same randomness r sent in step 2, computes

$$(\text{ek}, \text{ak}) = \text{Preprocess}(\text{crs}, ((\text{pk}_1, w_1), \dots, (\text{pk}_k, w_k)); r)$$

and responds with $\text{ct}^* \leftarrow \text{Enc}(\text{ek}, t, \text{msg}_b)$ to the adversary.

5. The adversary can make hint queries again of the form (i, ct) with $i \in [k] \setminus S$. If $\text{ct} = \text{ct}^*$, the experiment aborts and outputs a random $b' \xleftarrow{\mathcal{R}} \{0, 1\}$. Otherwise, the challenger responds with $\text{dh} \leftarrow \text{GetHint}(\text{crs}, \text{sk}_i, \text{ct})$.
6. The adversary outputs a bit b' which is the output of the experiment.

We say that Π_{PCTE} satisfies public-coin static security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

- **Succinctness:** There exist universal polynomials $p_{\text{ek}}(\lambda, \log k, L)$, $p_{\text{ct}}(\lambda, \log k, L, |\text{msg}|)$, and $p_{\text{dh}}(\lambda, \log k, L, |\text{msg}|)$ such that for all $\lambda, L \in \mathbb{N}$, all $k \in [2^\lambda]$, all weights $w_1, \dots, w_k \in [0, 2^L - 1]$, all $t \in [k \cdot (2^L - 1)]$, all messages $\text{msg} \in \mathcal{M}$, all crs in the support of $\text{Setup}(1^\lambda, 1^L)$, all $(\text{sk}_i, \text{pk}_i)$ in the support of $\text{KeyGen}(1^\lambda, 1^L, \text{crs})$ for $i \in [k]$, all (ek, ak) in the support of $\text{Preprocess}(\text{crs}, ((\text{pk}_1, w_1), \dots, (\text{pk}_k, w_k)))$, all ciphertexts ct in the support of $\text{Enc}(\text{ek}, t, \text{msg})$, all $i \in [k]$, and all dh_i in the support of $\text{GetHint}(\text{crs}, \text{sk}_i, \text{ct})$, we have

$$|\text{ek}| \leq \text{poly}(\lambda, \log k, L) \quad \text{and} \quad |\text{ct}|, |\text{dh}_i| \leq \text{poly}(\lambda, \log k, L, |\text{msg}|).$$

Remark 9.2 (Centralized and Silent Variants). Similar to puncturable public-coin threshold signatures, this definition implies a centralized weighted threshold encryption in the standard model, and a silent weighted threshold encryption in the random oracle model. See [Section 10](#) for more details.

9.1 Our Construction

In this section, we present our construction of public-coin weighted threshold encryption from witness encryption and puncturable public-coin weighted threshold signatures. In principle, the construction is simple: the key pair for each user is the key pair of the threshold signature scheme. Preprocessing a list of public key and weight pairs is done using the preprocessing algorithm of the signature scheme. Corrupted public keys are treated as adversarially chosen verification keys of that scheme; their secret keys are not needed by preprocessing. To encrypt a message m^* with a threshold t , we first sample a key pair $\text{sk}_{\text{OTS}}, \text{vk}_{\text{OTS}}$ for a regular (one-time) signature scheme. The one-time verification key vk_{OTS} is treated as the tag of the message m^* . Next, we encrypt m^* under witness encryption with respect to the relation that just checks an aggregate signature on the tag vk_{OTS} w.r.t. the same threshold. Finally, the witness encryption ciphertext is signed using sk_{OTS} . The ciphertext consists of the witness encryption ciphertext, the tag, and the signature. To decrypt, one needs to have enough signatures on the tag that can be aggregated to satisfy the threshold and thus satisfy the witness encryption relation and release the message. We refer to these individual signatures as decryption hints. Honest users only give out decryption hints on a tag vk_{OTS} if they are provided a valid signature on the accompanying witness encryption ciphertext. The security of this scheme relies on the fact that the threshold signature can be punctured such that no aggregate signature (over a certain threshold) exists for the challenge tag, which allows us to finally invoke witness encryption security and hide the challenge message.

Construction 9.3 (Public-Coin Weighted Threshold Encryption). Our construction of public-coin threshold encryption uses the following cryptographic ingredients:

- A strongly unforgeable one-time signature scheme $\Pi_{\text{OTS}} = (\text{OTS.Gen}, \text{OTS.Sign}, \text{OTS.Verify})$ as per [Definition 3.2](#). Let $\ell_{\text{vk}} := \ell_{\text{vk}}(\lambda)$ be the length of a verification key in the support of $\text{OTS.Gen}(1^\lambda)$.
- A puncturable public-coin weighted threshold signature scheme

$$\begin{aligned} \Pi_{\text{PCWTS}} = & (\text{PCWTS.Setup}, \text{PCWTS.KeyGen}, \text{PCWTS.Sign}, \text{PCWTS.PartialVerify}, \\ & \text{PCWTS.Preprocess}, \text{PCWTS.Aggregate}, \text{PCWTS.Verify}, \text{PCWTS.Sim}) \end{aligned}$$

as per [Definition 8.1](#), with message space $\{0, 1\}^{\ell_{\text{vk}}}$, so that the messages it signs are one-time verification keys. The randomness of [Definition 9.1](#) is spent entirely on PCWTS.Preprocess , so ℓ_r is the length of the randomness that algorithm consumes on a vector of k key and weight pairs, determined by crs and the weights before the corrupted public keys are supplied.

- A witness encryption scheme $\Pi_{\text{WE}} = (\text{WE.Enc}, \text{WE.Dec})$ as per [Definition 3.4](#).

Constants: An aggregated verification key vk_{agg} , a weight threshold t , and a one-time verification key $vk_{OTS} \in \{0, 1\}^{\ell_{vk}}$.
Inputs: An aggregate signature σ_{agg} .

1. Output $PPCWTS.Verify(vk_{agg}, t, vk_{OTS}, \sigma_{agg})$.

Figure 7: Circuit $C_{WE}[vk_{agg}, t, vk_{OTS}]$.

The algorithms for the scheme are defined as follows:

- **Setup**($1^\lambda, 1^L$): On input the security parameter λ and a bound L on the bit-length of the weights, the setup algorithm proceeds as follows:
 1. Sample and output $crs := crs_{PPCWTS} \leftarrow PPCWTS.Setup(1^\lambda, 1^L)$.
- **KeyGen**($1^\lambda, 1^L, crs$): On input the security parameter λ , a bound L on the bit-length of the weight, and the common reference string $crs = crs_{PPCWTS}$, the key-generation algorithm proceeds as follows:
 1. Sample $(vk, sk) \leftarrow PPCWTS.KeyGen(1^\lambda, 1^L, crs)$.
 2. Output the secret key sk and the public key $pk := vk$.
- **Preprocess**($crs, ((pk_1, w_1), \dots, (pk_k, w_k)); r$): On input the common reference string $crs = crs_{PPCWTS}$, a vector of $k \in [2^\lambda]$ public key and weight pairs (pk_i, w_i) where $pk_i = vk_i$ and $w_i \in [0, 2^L - 1]$, and randomness r , the preprocessing algorithm proceeds as follows:
 1. Compute $(vk_{agg}, ak) = PPCWTS.Preprocess(crs_{PPCWTS}, ((vk_1, w_1), \dots, (vk_k, w_k)); r)$.
 2. Output the encryption key $ek := (1^\lambda, vk_{agg})$ and the aggregation key ak of the signature scheme, which is what Dec needs in order to combine the parties' signatures.
- **Enc**(ek, t, msg): On input the encryption key $ek = (1^\lambda, vk_{agg})$, a weight threshold $t \in [k \cdot (2^L - 1)]$, and a message $msg \in \mathcal{M}$, the encryption algorithm proceeds as follows:
 1. Sample a one-time signature key pair $(vk_{OTS}, sk_{OTS}) \leftarrow OTS.Gen(1^\lambda)$.
 2. Construct the circuit $C_{WE} := C_{WE}[vk_{agg}, t, vk_{OTS}]$ from Figure 7, encrypt $ct_{WE} \leftarrow WE.Enc(1^\lambda, C_{WE}, msg)$ and sign $\sigma_{OTS} \leftarrow OTS.Sign(sk_{OTS}, ct_{WE})$.
 3. Output the ciphertext $ct := (vk_{OTS}, ct_{WE}, \sigma_{OTS})$.
- **GetHint**(crs, sk_i, ct): On input the common reference string crs , a secret key sk_i , and a ciphertext $ct = (vk_{OTS}, ct_{WE}, \sigma_{OTS})$, the hint algorithm proceeds as follows:
 1. Output $\perp$ if $OTS.Verify(vk_{OTS}, ct_{WE}, \sigma_{OTS}) \neq 1$.
 2. Output the decryption hint $dh_i \leftarrow PPCWTS.Sign(sk_i, vk_{OTS})$, a partial signature on the one-time verification key of the ciphertext.
- **Dec**($ak, \{(i, dh_i)\}_{i \in T}, ct$): On input the aggregation key ak , a set of indices $T \subseteq [k]$ together with the decryption hint $dh_i = \sigma_i$ of each party $i \in T$, and a ciphertext $ct = (vk_{OTS}, ct_{WE}, \sigma_{OTS})$, the decryption algorithm proceeds as follows:
 1. Output $\perp$ if $OTS.Verify(vk_{OTS}, ct_{WE}, \sigma_{OTS}) \neq 1$.
 2. Compute $\sigma_{agg} = PPCWTS.Aggregate(ak, vk_{OTS}, \{(i, \sigma_i)\}_{i \in T})$, so that each hint serves as the partial signature of its party on the message vk_{OTS} that the circuit of Figure 7 tests.
 3. Output $WE.Dec(ct_{WE}, \sigma_{agg})$.

Lemma 9.4 (Correctness). *Suppose that Π_{OTS} is correct (Definition 3.2), Π_{PPCWTS} satisfies signing correctness and aggregation correctness (Definition 8.1), and Π_{WE} is correct (Definition 3.4). Then the public-coin weighted threshold encryption scheme Π_{PCTE} of Construction 9.3 is correct.*

Proof. Fix $\lambda, L \in \mathbb{N}$, $k \in [2^\lambda]$, a message $\text{msg} \in \mathcal{M}$, weights $w_1, \dots, w_k \in [0, 2^L - 1]$, a threshold $t \in [k \cdot (2^L - 1)]$, and $T \subseteq [k]$ with $\sum_{i \in T} w_i \geq t$. Let $\text{crs} \leftarrow \text{Setup}(1^\lambda, 1^L)$, let $(\text{sk}_i, \text{pk}_i) \leftarrow \text{KeyGen}(1^\lambda, 1^L, \text{crs})$ for $i \in [k]$, let $(\text{ek}, \text{ak}) \leftarrow \text{Preprocess}(\text{crs}, ((\text{pk}_1, w_1), \dots, (\text{pk}_k, w_k)))$, let $\text{ct} \leftarrow \text{Enc}(\text{ek}, t, \text{msg})$, and let $\text{dh}_i \leftarrow \text{GetHint}(\text{crs}, \text{sk}_i, \text{ct})$ for $i \in T$.

By the construction:

- $\text{crs} = \text{crs}_{\text{PPCWTS}} \leftarrow \text{PPCWTS.Setup}(1^\lambda, 1^L)$;
- $\text{pk}_i = \text{vk}_i$, where $(\text{vk}_i, \text{sk}_i) \leftarrow \text{PPCWTS.KeyGen}(1^\lambda, 1^L, \text{crs})$, for each $i \in [k]$;
- $\text{ek} = (1^\lambda, \text{vk}_{\text{agg}})$, where $(\text{vk}_{\text{agg}}, \text{ak}) \leftarrow \text{PPCWTS.Preprocess}(\text{crs}_{\text{PPCWTS}}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)))$;
- $\text{ct} = (\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$, where $(\text{vk}_{\text{OTS}}, \text{sk}_{\text{OTS}}) \leftarrow \text{OTS.Gen}(1^\lambda)$, $\text{ct}_{\text{WE}} \leftarrow \text{WE.Enc}(1^\lambda, C_{\text{WE}}[\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}], \text{msg})$ and $\sigma_{\text{OTS}} \leftarrow \text{OTS.Sign}(\text{sk}_{\text{OTS}}, \text{ct}_{\text{WE}})$.

By correctness of Π_{OTS} , $\text{OTS.Verify}(\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) = 1$. Hence neither GetHint nor Dec outputs $\perp$, and each $\text{dh}_i = \sigma_i$ lies in the support of $\text{PPCWTS.Sign}(\text{sk}_i, \text{vk}_{\text{OTS}})$ for $i \in T$. By signing correctness of Π_{PPCWTS} , we have $\text{PPCWTS.PartialVerify}(\text{vk}_i, \text{vk}_{\text{OTS}}, \sigma_i) = 1$ for every $i \in T$. By aggregation correctness of Π_{PPCWTS} , applied to the threshold t and the set T , the signature $\sigma_{\text{agg}} = \text{PPCWTS.Aggregate}(\text{ak}, \text{vk}_{\text{OTS}}, \{(i, \sigma_i)\}_{i \in T})$ that Dec computes satisfies $\text{PPCWTS.Verify}(\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}, \sigma_{\text{agg}}) = 1$, that is, $C_{\text{WE}}[\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}](\sigma_{\text{agg}}) = 1$ for the circuit of Figure 7. By correctness of Π_{WE} , $\text{Dec}(\text{ak}, \{(i, \text{dh}_i)\}_{i \in T}, \text{ct}) = \text{WE.Dec}(\text{ct}_{\text{WE}}, \sigma_{\text{agg}}) = \text{msg}$. $\square$

Lemma 9.5 (Succinctness). *Suppose that Π_{PPCWTS} is succinct (Definition 8.1), witnessed by a polynomial $\text{poly}_{\text{PPCWTS}}(\cdot, \cdot, \cdot)$, and that Π_{PPCWTS} , Π_{OTS} and Π_{WE} run in polynomial time. Then the public-coin weighted threshold encryption scheme Π_{PCTE} of Construction 9.3 is succinct.*

Proof. Fix $\lambda, L, k, \text{msg}$, the weights $w_1, \dots, w_k$ and the threshold t as above, and let crs , the keys $(\text{sk}_i, \text{pk}_i)$, the pair (ek, ak) , the ciphertext $\text{ct} = (\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$ and the hints dh_i lie in the supports named in Definition 9.1.

By construction, $\text{ek} = (1^\lambda, \text{vk}_{\text{agg}})$ for any aggregate verification key vk_{agg} in the support of the preprocessing algorithm $\text{PPCWTS.Preprocess}(\text{crs}_{\text{PPCWTS}}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)))$, so succinctness of Π_{PPCWTS} gives $|\text{ek}| = \lambda + |\text{vk}_{\text{agg}}| \leq \lambda + \text{poly}_{\text{PPCWTS}}(\lambda, \log k, L)$.

By construction, dh_i is in the support of $\text{PPCWTS.Sign}(\text{sk}_i, \text{vk}_{\text{OTS}})$, where sk_i is output by the honest key-generation algorithm $\text{PPCWTS.KeyGen}(1^\lambda, 1^L, \text{crs})$ and $|\text{vk}_{\text{OTS}}| = \ell_{\text{vk}} \leq \text{poly}(\lambda)$. Neither algorithm receives k in any form and both are efficient, so $|\text{dh}_i| \leq \text{poly}(\lambda, L)$.

By the construction, ct_{WE} is in the support of $\text{WE.Enc}(1^\lambda, C_{\text{WE}}, \text{msg})$ for the circuit $C_{\text{WE}} := C_{\text{WE}}[\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}]$ of Figure 7. Since $t \in [k \cdot (2^L - 1)]$ has $|t| \leq L + \lceil \log k \rceil$, the hardwired values $(\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}})$ have total length $\text{poly}(\lambda, \log k, L)$. Succinctness of Π_{PPCWTS} bounds the aggregate signature length by $\text{poly}(\lambda, \log k, L)$, and PPCWTS.Verify is efficient, so $|C_{\text{WE}}| \leq \text{poly}(\lambda, \log k, L)$. Since WE.Enc runs in time polynomial in the length of its inputs, $|\text{ct}_{\text{WE}}| \leq \text{poly}(\lambda + |C_{\text{WE}}| + |\text{msg}|) \leq \text{poly}(\lambda, \log k, L, |\text{msg}|)$. Likewise OTS.Sign runs in time polynomial in $|\text{sk}_{\text{OTS}}| + |\text{ct}_{\text{WE}}|$, so $|\sigma_{\text{OTS}}| \leq \text{poly}(\lambda, \log k, L, |\text{msg}|)$, and hence $|\text{ct}| = \ell_{\text{vk}} + |\text{ct}_{\text{WE}}| + |\sigma_{\text{OTS}}| \leq \text{poly}(\lambda, \log k, L, |\text{msg}|)$. $\square$

Theorem 9.6 (Public-Coin Static Security). *Suppose that Π_{OTS} satisfies strong one-time unforgeability (Definition 3.2), Π_{PPCWTS} satisfies mode indistinguishability and punctured security (Definition 8.1), and Π_{WE} is message hiding (Definition 3.4). Then Construction 9.3 satisfies static security.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the static security game against Construction 9.3. For a bit $b \in \{0, 1\}$ we define the following experiments; each is described as a modification of the previous one, and all of them output $\mathcal{A}$'s guess b' . In every experiment and reduction below, once the challenge ciphertext ct^* has been generated, any hint query with $\text{ct} = \text{ct}^*$ causes an immediate halt with a uniformly random output bit, as in Definition 9.1.

- $\text{Real}^{(b)}$: the static security game of Definition 9.1 with challenge bit b , with the algorithms of Construction 9.3 written out. In full:

1. The adversary $\mathcal{A}(1^\lambda)$ outputs $1^L, 1^k$, weights $w_1, \dots, w_k \in [0, 2^L - 1]$, and a corruption set $S \subseteq [k]$.
2. The challenger samples the following:
 - A common reference string $\text{crs}_{\text{PPCWTS}} \leftarrow \text{PPCWTS.Setup}(1^\lambda, 1^L)$.
 - For each $i \in [k] \setminus S$: a key pair $(\text{vk}_i, \text{sk}_i) \leftarrow \text{PPCWTS.KeyGen}(1^\lambda, 1^L, \text{crs}_{\text{PPCWTS}})$.
 - A uniform random string $r \xleftarrow{\mathcal{R}} \{0, 1\}^{\ell_r}$, where ℓ_r is determined by the public parameters and weights, independently of the public-key values.

The challenger sends $\text{crs}_{\text{PPCWTS}}$, r , and $(\text{vk}_i)_{i \in [k] \setminus S}$ to $\mathcal{A}$.

3. Throughout the remainder of the experiment, $\mathcal{A}$ may make hint queries (i, ct) with $i \in [k] \setminus S$. The challenger parses $\text{ct} = (\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$. If $\text{OTS.Verify}(\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) \neq 1$, it replies with $\perp$. Otherwise it replies with $\text{PPCWTS.Sign}(\text{sk}_i, \text{vk}_{\text{OTS}})$.
4. The adversary $\mathcal{A}$ outputs arbitrary corrupted public keys $(\text{pk}_i)_{i \in S}$, which the challenger identifies with $(\text{vk}_i)_{i \in S}$, a threshold $t \in [k \cdot (2^L - 1)]$ with $t > \sum_{i \in S} w_i$ and two messages $\text{msg}_0, \text{msg}_1 \in \mathcal{M}$ of the same length.
5. The challenger combines the honest verification keys with those supplied by $\mathcal{A}$ and computes

$$(\text{vk}_{\text{agg}}, \text{ak}) = \text{PPCWTS.Preprocess}(\text{crs}_{\text{PPCWTS}}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)); r).$$

It samples $(\text{vk}_{\text{OTS}}^*, \text{sk}_{\text{OTS}}^*) \leftarrow \text{OTS.Gen}(1^\lambda)$, computes $\text{ct}_{\text{WE}}^* \leftarrow \text{WE.Enc}(1^\lambda, C_{\text{WE}}[\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*], \text{msg}_b)$ and $\sigma_{\text{OTS}}^* \leftarrow \text{OTS.Sign}(\text{sk}_{\text{OTS}}^*, \text{ct}_{\text{WE}}^*)$, and sends $\text{ct}^* = (\text{vk}_{\text{OTS}}^*, \text{ct}_{\text{WE}}^*, \sigma_{\text{OTS}}^*)$ to $\mathcal{A}$.

6. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.
- $\text{Hyb}_1^{(b)}$: same as $\text{Real}^{(b)}$, except that the one-time key pair of the challenge ciphertext is sampled before the setup and that the experiment aborts if $\mathcal{A}$ ever asks an honest party to sign under it. Namely, **the challenger samples $(\text{vk}_{\text{OTS}}^*, \text{sk}_{\text{OTS}}^*) \leftarrow \text{OTS.Gen}(1^\lambda)$ before step 2** and uses it in step 5, and step 3 becomes:
 3. Throughout the remainder of the experiment, $\mathcal{A}$ may make hint queries (i, ct) with $i \in [k] \setminus S$, answered as in $\text{Real}^{(b)}$, except that **the experiment halts and outputs a random bit $b' \xleftarrow{\mathcal{R}} \{0, 1\}$ if ct parses as $(\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$ with $\text{vk}_{\text{OTS}} = \text{vk}_{\text{OTS}}^*$ and $\text{OTS.Verify}(\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) = 1$** . The halt agrees with the convention of [Definition 9.1](#) on a query with $\text{ct} = \text{ct}^*$, which triggers both rules.

In this and the later hybrids, every hint query of an honest party that is answered with something other than $\perp$ carries $\text{vk}_{\text{OTS}} \neq \text{vk}_{\text{OTS}}^*$.

- $\text{Hyb}_2^{(b)}$: same as $\text{Hyb}_1^{(b)}$, except that the common reference string, the honest key pairs and the preprocessing randomness are simulated. Namely step 2 is changed as follows:

2. The challenger samples

$$\begin{aligned} & (\text{crs}_{\text{PPCWTS}}, ((\text{vk}_i, \text{sk}_i))_{i \in [k] \setminus S}, r) \\ & \leftarrow \text{PPCWTS.Sim}(1^\lambda, 1^L, 1^k, \{w_i\}_{i \in [k]}, S, \text{vk}_{\text{OTS}}^*). \end{aligned}$$

It sends $\text{crs}_{\text{PPCWTS}}$, r , and $(\text{vk}_i)_{i \in [k] \setminus S}$ to $\mathcal{A}$, retaining the honest signing keys to answer hint queries.

Step 5 uses this same simulated randomness r to compute $(\text{vk}_{\text{agg}}, \text{ak})$.

- $\text{Hyb}_3^{(b)}$: same as $\text{Hyb}_2^{(b)}$, except that the experiment gives up when the circuit of the challenge ciphertext is satisfiable. Namely step 5 is changed as follows:
 5. As in $\text{Hyb}_2^{(b)}$, except that **once vk_{agg} and t are fixed, the experiment halts and outputs a random bit $b' \xleftarrow{\mathcal{R}} \{0, 1\}$ if there exists a σ_{agg} with $\text{PPCWTS.Verify}(\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*, \sigma_{\text{agg}}) = 1$** , and only otherwise sends ct^* to $\mathcal{A}$.

Note that this condition is not efficiently checkable, so the challenger of $\text{Hyb}_3^{(b)}$ is not efficient.

Claim 9.7. Assume Π_{OTS} satisfies strong one-time unforgeability. Then for any efficient $\mathcal{A}$ and any $b \in \{0, 1\}$ there exists a negligible negl such that

$$\left| \Pr[\text{Real}^{(b)}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_1^{(b)}(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. The pair $(\text{vk}_{\text{OTS}}^*, \text{sk}_{\text{OTS}}^*)$ has the same distribution in both experiments and is used only in step 5, so moving its sampling does not change $\mathcal{A}$'s view. Write E for the event that $\mathcal{A}$ makes a hint query (i, ct) with $i \in [k] \setminus S$ and $\text{ct} = (\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$ such that $\text{vk}_{\text{OTS}} = \text{vk}_{\text{OTS}}^*$ and $\text{OTS.Verify}(\text{vk}_{\text{OTS}}^*, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) = 1$, and let ct be the first such query. The two experiments are identical unless E occurs, and on $E \wedge [\text{ct} = \text{ct}^*]$ both output a random bit. Hence the claimed difference is at most $\Pr[E \wedge \text{ct} \neq \text{ct}^*]$. Consider the following adversary $\mathcal{B}$ for the game of Definition 3.2.

1. Receive vk and set $\text{vk}_{\text{OTS}}^* := \text{vk}$.
2. Run $\text{Hyb}_1^{(b)}$ with this vk_{OTS}^* , sampling everything else itself and answering hint queries as that experiment does.
3. At step 5, submit ct_{WE}^* as the single signing query to obtain σ_{OTS}^* .
4. On the first hint query satisfying E with $\text{ct} \neq \text{ct}^*$, halt and output $(\text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$.

Algorithm $\mathcal{B}$ never needs sk_{OTS}^* , and its simulation is perfect up to the halt. The pair it outputs verifies under vk_{OTS}^* , and $\text{ct} \neq \text{ct}^*$ with both carrying vk_{OTS}^* gives $(\text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) \neq (\text{ct}_{\text{WE}}^*, \sigma_{\text{OTS}}^*)$; if the query precedes the challenge then $\mathcal{B}$ has made no signing query and any verifying pair wins. Hence $\Pr[E \wedge \text{ct} \neq \text{ct}^*] \leq \text{negl}(\lambda)$. $\square$

Claim 9.8. Assume Π_{PPCWTs} satisfies mode indistinguishability. Then for any efficient $\mathcal{A}$ and any $b \in \{0, 1\}$ there exists a negligible negl such that

$$\left| \Pr[\text{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_2^{(b)}(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. Consider the following adversary $\mathcal{B}$ for the mode indistinguishability game of Definition 8.1.

1. Run $\mathcal{A}(1^\lambda)$ to obtain $1^L, 1^k, \{w_i\}_{i \in [k]}, S$, sample $(\text{vk}_{\text{OTS}}^*, \text{sk}_{\text{OTS}}^*) \leftarrow \text{OTS.Gen}(1^\lambda)$, and send $1^L, 1^k, \{w_i\}_{i \in [k]}, S$ and $m^* := \text{vk}_{\text{OTS}}^*$ to the challenger.
2. Receive $\text{crs}_{\text{PPCWTs}}, (\text{vk}_i)_{i \in [k] \setminus S}$ and r , and forward them to $\mathcal{A}$.
3. On a hint query (i, ct) with $i \in [k] \setminus S$, parse $\text{ct} = (\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$ and halt with a random bit if $\text{vk}_{\text{OTS}} = \text{vk}_{\text{OTS}}^*$ and $\text{OTS.Verify}(\text{vk}_{\text{OTS}}^*, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) = 1$; otherwise reply with $\perp$ if $\text{OTS.Verify}(\text{vk}_{\text{OTS}}, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}}) \neq 1$, and otherwise forward $(i, \text{vk}_{\text{OTS}})$ to the signing oracle.
4. When $\mathcal{A}$ outputs $(\text{pk}_i)_{i \in S}, t, \text{msg}_0, \text{msg}_1$, set $\text{vk}_i := \text{pk}_i$ for each $i \in S$ and compute

$$(\text{vk}_{\text{agg}}, \text{ak}) = \text{PPCWTs.Preprocess}(\text{crs}_{\text{PPCWTs}}, ((\text{vk}_1, w_1), \dots, (\text{vk}_k, w_k)); r).$$

Form ct^* as in step 5 with msg_b , send ct^* to $\mathcal{A}$, and output $\mathcal{A}$'s guess.

The forwarded queries are legal: they carry an index $i \in [k] \setminus S$ and, by the halt of $\text{Hyb}_1^{(b)}$, a message $\text{vk}_{\text{OTS}} \neq \text{vk}_{\text{OTS}}^* = m^*$. In mode $b_{\text{PPCWTs}} = 0$ the challenger draws $\text{crs}_{\text{PPCWTs}}$ and the keys from PPCWTs.Setup and PPCWTs.KeyGen and r uniformly, so $\mathcal{B}$ simulates $\text{Hyb}_1^{(b)}$. Conversely, in mode $b_{\text{PPCWTs}} = 1$, it samples them via the simulator $\text{PPCWTs.Sim}(1^\lambda, 1^L, 1^k, \{w_i\}_{i \in [k]}, S, \text{vk}_{\text{OTS}}^*)$, so $\mathcal{B}$ simulates $\text{Hyb}_2^{(b)}$. Its advantage is the claimed difference. $\square$

Claim 9.9. Assume Π_{PPCWTs} satisfies punctured security. Then for any efficient $\mathcal{A}$ and any $b \in \{0, 1\}$ there exists a negligible negl such that

$$\left| \Pr[\text{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3^{(b)}(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. Write E for the event that some σ_{agg} satisfies $\text{PPCWTS.Verify}(\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*, \sigma_{\text{agg}}) = 1$. The two experiments are coupled until the halting rule is evaluated, so the claimed difference is at most $\Pr[E]$. Consider the following unbounded adversary $\mathcal{B}$ for the punctured security game of [Definition 8.1](#).

1. Run $\mathcal{A}(1^\lambda)$ to obtain $1^L, 1^k, \{w_i\}_{i \in [k]}, S$, sample $(\text{vk}_{\text{OTS}}^*, \text{sk}_{\text{OTS}}^*) \leftarrow \text{OTS.Gen}(1^\lambda)$, and send $1^k, \{w_i\}_{i \in [k]}, S$ and $m^* := \text{vk}_{\text{OTS}}^*$ to the challenger.
2. Receive $\text{crs}_{\text{PPCWTS}}, (\text{vk}_i)_{i \in [k] \setminus S}$ and r , forward them to $\mathcal{A}$, and answer its hint queries as in the third step of the reduction of [Claim 9.8](#).
3. When $\mathcal{A}$ outputs $(\text{pk}_i)_{i \in S}, t, \text{msg}_0, \text{msg}_1$, set $\text{vk}_i := \text{pk}_i$ for each $i \in S$ and compute vk_{agg} from $\text{crs}_{\text{PPCWTS}}$, the keys and r as in step 5, search for a σ_{agg} with $\text{PPCWTS.Verify}(\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*, \sigma_{\text{agg}}) = 1$, and output the corrupted verification keys $(\text{vk}_i)_{i \in S}$, the threshold t , and the forgery σ_{agg} .

The simulation is perfect: step 2 of the game uses the same call to PPCWTS.Sim as $\text{Hyb}_2^{(b)}$ to generate $\text{crs}_{\text{PPCWTS}}$, the honest key pairs, and r , and $\mathcal{B}$ samples the rest itself. Both preprocessing calls use the same corrupted verification keys supplied by $\mathcal{A}$, even though these keys may depend on the honest keys, r , and the earlier hint responses. Since $\mathcal{B}$ is unbounded, it can search all inputs of the circuit $C_{\text{WE}}[\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*]$ in step 3 and find an accepting aggregate signature σ_{agg} if one exists. The output of $\mathcal{B}$ is legal, as $t \in [k \cdot (2^L - 1)]$ and $t > \sum_{i \in S} w_i$. Algorithm $\mathcal{B}$ wins the punctured security game exactly if the signature it submits satisfies $\text{PPCWTS.Verify}(\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*, \sigma_{\text{agg}}) = 1$, so $\mathcal{B}$ wins whenever E occurs. By the punctured security of Π_{PPCWTS} , we conclude that $\Pr[E] \leq \text{negl}(\lambda)$ and the claim follows. $\square$

Claim 9.10. Assume Π_{WE} is message hiding. Then for any efficient $\mathcal{A}$ there exists a negligible negl such that

$$\left| \Pr[\text{Hyb}_3^{(0)}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3^{(1)}(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda).$$

Proof. Consider the following adversary $\mathcal{B}$ for the message hiding game of [Definition 3.4](#).

1. Run $\mathcal{A}(1^\lambda)$ to obtain $1^L, 1^k, \{w_i\}_{i \in [k]}, S$, sample $(\text{vk}_{\text{OTS}}^*, \text{sk}_{\text{OTS}}^*) \leftarrow \text{OTS.Gen}(1^\lambda)$ and

$$\begin{aligned} & (\text{crs}_{\text{PPCWTS}}, ((\text{vk}_i, \text{sk}_i))_{i \in [k] \setminus S}, r) \\ & \leftarrow \text{PPCWTS.Sim}(1^\lambda, 1^L, 1^k, \{w_i\}_{i \in [k]}, S, \text{vk}_{\text{OTS}}^*). \end{aligned}$$

Send $\text{crs}_{\text{PPCWTS}}, r$, and $(\text{vk}_i)_{i \in [k] \setminus S}$ to $\mathcal{A}$.

2. Answer hint queries as $\text{Hyb}_3^{(b)}$ does, using the sk_i it holds for every honest party $i \in [k] \setminus S$.
3. When $\mathcal{A}$ outputs $(\text{pk}_i)_{i \in S}, t, \text{msg}_0, \text{msg}_1$, set $\text{vk}_i := \text{pk}_i$ for each $i \in S$ and compute vk_{agg} from $\text{crs}_{\text{PPCWTS}}$, the keys and r as in step 5, and send the circuit $C_{\text{WE}}^* := C_{\text{WE}}[\text{vk}_{\text{agg}}, t, \text{vk}_{\text{OTS}}^*]$ and $\text{msg}_0, \text{msg}_1$ to the challenger.
4. On receiving ct_{WE} , compute $\sigma_{\text{OTS}} \leftarrow \text{OTS.Sign}(\text{sk}_{\text{OTS}}^*, \text{ct}_{\text{WE}})$, send $\text{ct}^* := (\text{vk}_{\text{OTS}}^*, \text{ct}_{\text{WE}}, \sigma_{\text{OTS}})$ to $\mathcal{A}$, keep answering hint queries, and output $\mathcal{A}$'s guess.

When the Π_{WE} challenger encrypts message msg_b , the algorithm $\mathcal{B}$ perfectly simulates $\text{Hyb}_3^{(b)}(\mathcal{A})$. Moreover, algorithm $\mathcal{B}$ is efficient and never evaluates the halting condition of $\text{Hyb}_3^{(b)}$, since its challenger aborts with a random bit precisely when C_{WE}^* is satisfiable, which is that halting rule. The claim now follows by the message hiding of Π_{WE} . $\square$

Combining the claims, $\left| \Pr[\text{Real}^{(0)}(\mathcal{A}) = 1] - \Pr[\text{Real}^{(1)}(\mathcal{A}) = 1] \right| \leq \text{negl}(\lambda)$, which is static security. $\square$

10 Silent Setup in the Random Oracle Model

Weighted threshold encryption with silent setup. We sketch how to obtain a weighted threshold encryption scheme with silent setup [GKPW24] from any public-coin weighted threshold encryption scheme as per Definition 9.1. The idea is to use a random oracle to derive the coins of Preprocess from its public input. Namely, for any common reference string crs and list of key-weight pairs $V := ((pk_1, w_1), \dots, (pk_k, w_k))$, let $r = \mathcal{H}(\text{crs} \circ V)$ where $\mathcal{H}$ is a hash function modeled as a random oracle returning the number of random bits required by Preprocess on this input. Define

$$\text{Preprocess}^{\mathcal{H}}(\text{crs}, V) = \text{Preprocess}(\text{crs}, V; \mathcal{H}(\text{crs} \circ V)).$$

Thus, relative to the oracle, preprocessing is deterministic: anyone with the same public keys and weights can independently compute the same encryption and aggregation keys, without interaction or an additional source of shared randomness. The common reference string and independent key generation are as before, as are encryption, hint generation, and decryption. In particular, the oracle input does not depend on the encryption threshold or message, so the resulting keys can be reused for different thresholds and messages.

Security proof sketch. The public-coin requirement is useful precisely because security must hold even when the adversary sees the preprocessing coins. In the random oracle model, the adversary learns these coins by querying the challenger for the evaluation of $\mathcal{H}$ on the public preprocessing input.

Let $\mathcal{A}$ be an adversary against the silent scheme. The reduction forwards $\mathcal{A}$'s initial choice of 1^L , 1^k , weights $w_1, \dots, w_k$, and corruption set S to its challenger in the static security game of Definition 9.1. It receives crs , the preprocessing randomness r , and only the honest public keys $(pk_i)_{i \in [k] \setminus S}$, and gives crs and those public keys to $\mathcal{A}$. The adversary may choose arbitrary corrupted public keys, so the reduction does not yet know the complete challenge vector V^* . Let $q_{\mathcal{H}}$ be a polynomial bound on the number of oracle queries before the challenge. The reduction guesses an index $J \xleftarrow{R} [q_{\mathcal{H}} + 1]$ among the distinct oracle inputs of the form $\text{crs} \circ V$ that agree with the fixed weights and honest public keys, counting the challenge input as one additional input if it has not already been queried. On the J th such fresh input it returns r ; on every other fresh input it returns an independent uniform string of the required length. It stores all answers for repeated queries. The prescribed randomness length depends only on crs and the weights, so r has the correct length for every candidate vector, regardless of its corrupted keys.

The reduction forwards hint queries to its challenger and returns the resulting decryption hints. When $\mathcal{A}$ supplies its corrupted public keys, challenge threshold t , and equal-length messages $\text{msg}_0, \text{msg}_1$, the reduction forms V^* from those keys and the honest public keys. If $\text{crs} \circ V^*$ is fresh, it assigns its oracle answer using the same rule. If this is not the input selected by J , the reduction aborts with a uniformly random output bit. Otherwise, $\mathcal{H}(\text{crs} \circ V^*) = r$, and the reduction forwards the corrupted public keys, t , and $\text{msg}_0, \text{msg}_1$ to its challenger and returns the challenge ciphertext ct^* . The challenger uses the same randomness r that it sent at the start, so its encryption key is precisely the one obtained from $\text{Preprocess}^{\mathcal{H}}(\text{crs}, V^*)$. The reduction continues to relay hint queries, subject to the same restriction on querying ct^* , and finally outputs $\mathcal{A}$'s guess.

The programmed oracle answer has the correct distribution because r is sampled uniformly and independently of the common reference string and honest keys; corrupted keys may depend on oracle answers. All preprocessing computations on the challenge vector therefore use the same randomness as the challenger, even when $\mathcal{A}$ computes the encryption key before choosing its challenge messages. Before the challenge, the simulated view has the same distribution for every guess J , since every fresh oracle answer is uniform. The guess therefore selects the challenge input with probability $1/(q_{\mathcal{H}} + 1)$. Conditioned on this correct guess, the reduction perfectly simulates the silent security game; on an incorrect guess, its output is independent of the challenge bit. Static security of the public-coin scheme thus implies static security of the silent scheme in the random oracle model, with a polynomial loss of $q_{\mathcal{H}} + 1$.

Weighted (puncturable) threshold signatures with silent setup. We can similarly extend the idea to public-coin (puncturable) threshold signatures: similar to the threshold encryption case, the randomness for the preprocessing algorithm is derived from the random oracle. The reduction again uses the randomness from its public-coin challenger in order to program the random oracle on the challenge. Note that for the puncturable version, there are two security properties: (public-coin) mode indistinguishability and puncturable security, both of which can be handled the same way.

Acknowledgments

Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator Award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, an Amazon Research Award, and a gift from the Stellar Development Foundation.

References

- [ABD⁺26] Amit Agarwal, Kushal Babel, Sourav Das, Babak Poorebrahim Gilkalaye, Arup Mondal, Benny Pinkas, Peter Rindal, and Aayush Yadav. Weighted batched threshold encryption with applications to mempool privacy. In *IEEE S&P*, 2026.
- [AFMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In *ASIACRYPT*, 2020.
- [AGY26] Abtin Afshar, Rishab Goyal, and Saikumar Yadugiri. Silent distributed cryptography for DNFs and threshold policies from lattices. *IACR Cryptol. ePrint Arch.*, 2026.
- [BBK⁺23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, 2023.
- [BCI⁺13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013.
- [BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In *EUROCRYPT*, 2024.
- [BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, 2010.
- [BDS25] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. Rate-1 statistical non-interactive zero-knowledge. In *CRYPTO*, 2025.
- [BFN26] David Balbás, Dario Fiore, and Duy Nguyen. Splitting bilinear groups: New translations from composite-to prime-order with applications to batch arguments for NP. In *CRYPTO*, 2026.
- [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 *CRYPTO*, 2001.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.
- [BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, 2017.
- [BHZ87] Ravi B. Boppana, Johan Håstad, and Stathis Zachos. Does co-NP have short interactive proofs? *Inf. Process. Lett.*, 25(2), 1987.
- [BKP⁺24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron D Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. In *STOC*, 2024.
- [Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, 2012.
- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, 2011.

- [CCH⁺19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N Rothblum, Ron D Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In *STOC*, 2019.
- [CEW25] Binyi Chen, Noel Elias, and David J Wu. Pairing-based batch arguments for NP with a linear-size CRS. In *ASIACRYPT*, 2025.
- [CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, 1998.
- [CGJ⁺23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In *CRYPTO*, 2023.
- [CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, 1997.
- [CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO*, 2021.
- [CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, 2021.
- [CWY26] Jeffrey Champion, David J. Wu, and Shota Yamada. Silent threshold encryption from lattices. *IACR Cryptol. ePrint Arch.*, 2026.
- [DCX⁺23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In *ACM CCS*, 2023.
- [DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *STOC*, 1991.
- [Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO*, 1987.
- [Des92] Yvo Desmedt. Threshold cryptosystems. In *international workshop on the theory and application of cryptographic techniques*, 1992.
- [DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *FOCS*, 2022.
- [DHJQ26] Jesko Dujmovic, Yao-Ching Hsieh, Abhishek Jain, and Willy Quach. How to use polynomially-hard iO: Turing machine obfuscation and more. In *CRYPTO*, 2026.
- [DJJ⁺25] Pratish Datta, Abhishek Jain, Zhengzhong Jin, Alexis Korb, Surya Mathialagan, and Amit Sahai. Incrementally verifiable computation for NP from standard assumptions. In *CRYPTO*, 2025.
- [DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J Wu. Succinct witness encryption for batch languages and applications. In *ASIACRYPT*, 2025.
- [FPRT26] Alexander Frolov, Aditi Partap, Max Resnick, and Ertem Nusret Tas. Weighted batch threshold encryption with efficient DKG. *IACR Cryptol. ePrint Arch.*, 2026. Available at <https://eprint.iacr.org/2026/2053>.
- [Fra89] Yair Frankel. A practical protocol for large group oriented networks. In *EUROCRYPT*, 1989.
- [FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.
- [FY53] Ronald Aylmer Fisher and Frank Yates. *Statistical tables for biological, agricultural and medical research*. Oliver and Boyd, 1953.
- [Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009.

- [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 *FOCS*, 2013.
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.
- [GGN10] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM Journal on Computing*, 39(7), 2010.
- [GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.
- [GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4), 1998.
- [GJM⁺23] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: MPC, encryption and signatures. In *CRYPTO*, 2023.
- [GJM⁺24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *IEEE S&P*, 2024.
- [GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.
- [GSW26] Aarushi Goel, Swagata Sasmal, and Mingyuan Wang. Weighted cryptography with weight-independent complexity. In *CRYPTO*, 2026.
- [GVW01] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. In *ICALP*, 2001.
- [GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *PKC*, 2019.
- [GY26] Rishab Goyal and Saikumar Yadugiri. Distributed monotone policy encryption with stronger security for DNFs and threshold policies from lattices. *IACR Cryptol. ePrint Arch.*, 2026.
- [HLR21] Justin Holmgren, Alex Lombardi, and Ron D Rothblum. Fiat–Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In *STOC*, 2021.
- [HSY26] Ryuya Hayashi, Yusuke Sakai, and Shota Yamada. Attribute-based signatures for circuits with optimal parameter size from standard assumptions. In *PKC*, 2026.
- [HW15] Pavel Hubáček and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.
- [HW26] Justin Holmgren and Brent Waters. Separating selective opening security from standard security, assuming indistinguishability obfuscation. In *PKC*, 2026.
- [KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, 2023.
- [KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, 2019.
- [Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, 1989.

- [MRV⁺21] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *IEEE S&P*, 2021.
- [Nas26] Shafik Nassar. The power of rerandomization in Obfustopia: Collision-resistant hash, somewhere-extractable BARGs, and more. In *TCC*, 2026.
- [NWW24] Shafik Nassar, Brent Waters, and David J Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. In *TCC*, 2024.
- [NWW25] Shafik Nassar, Brent Waters, and David J Wu. Monotone-policy BARGs and more from BARGs and quadratic residuosity. In *PKC*, 2025.
- [NWW26] Shafik Nassar, Brent Waters, and David J. Wu. Obfuscating non-evasive programs: Making witness encryption positional. *IACR Cryptol. ePrint Arch.*, 2026. Available at <https://eprint.iacr.org/2026/1932>.
- [Pol26a] Guru-Vamsi Policharla. DKG is all you need. *IACR Cryptol. ePrint Arch.*, 2026. Available at <https://eprint.iacr.org/2026/2205>.
- [Pol26b] Guru-Vamsi Policharla. Weighted threshold encryption with silent setup. *IACR Cryptol. ePrint Arch.*, 2026. Available at <https://eprint.iacr.org/2026/2148>.
- [PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *FOCS*, 2022.
- [PP26] Omer Paneth and Rafael Pass. Verifiable divide-and-conquer. *IACR Cryptol. ePrint Arch.*, 2026. Available at <https://eprint.iacr.org/2026/662>.
- [RSY21] Leonid Reyzin, Adam D. Smith, and Sophia Yakoubov. Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In *Cyber Security Cryptography and Machine Learning*, 2021.
- [Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.
- [Wee05] Hoeteck Wee. On round-efficient argument systems. In *ICALP*, 2005.
- [WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, 2022.
- [WW26] Brent Waters and David J. Wu. Silent threshold cryptography from pairings: Expressive policies in the plain model. In *EUROCRYPT*, 2026.

A Dynamic Somewhere-Extractable Hash Construction

In this section, we construct a dynamic somewhere-extractable hash function as per [Definition 4.1](#). The construction follows that of Hubáček and Wichs [\[HW15\]](#): the blocks of the vector are hashed using a Merkle tree, where an FHE encryption of some index i^* dictates which block should be homomorphically propagated to the root. The distinction is that this bound index i^* is derived by applying an FHE encrypted function f to an input tag τ . The function is given at setup time and is encoded in the public parameters, but the tag is given at hashing time (equivalently, we separate the hashing algorithm into two parts: one that *deterministically* generates the hash from the public parameters and the tag, and the other is the standard Merkle hashing algorithm). This way, the tag dynamically determines the bound index.

A.1 Building Block

Definition A.1 (Leveled Homomorphic Encryption [Gen09]). A *leveled homomorphic encryption* scheme over message space $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\text{LHE}} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ with the following syntax:

- $\text{KeyGen}(1^\lambda, 1^d) \rightarrow (\text{sk}, \text{pk})$: On input a security parameter $\lambda \in \mathbb{N}$ and a depth bound $d \in \mathbb{N}$, the key-generation algorithm outputs a secret key sk and a public key pk .
- $\text{Enc}(\text{pk}, \text{msg}) \rightarrow \text{ct}$: On input a public key pk and a message $\text{msg} \in X_\lambda$, the encryption algorithm outputs a ciphertext ct .
- $\text{Dec}(\text{sk}, \text{ct}) \rightarrow \text{msg} \in X_\lambda \cup \{\perp\}$: On input a secret key sk and a ciphertext ct , the decryption algorithm either outputs a plaintext $\text{msg} \in X_\lambda$ or a special symbol $\text{msg} = \perp$. The decryption algorithm is deterministic.
- $\text{Eval}(\text{pk}, C, \text{ct}) \rightarrow \text{ct}'$: On input a public key pk , a Boolean circuit C , and a ciphertext ct , the homomorphic evaluation algorithm outputs a new ciphertext ct' . The evaluation algorithm is *deterministic*. We extend Eval to take multiple ciphertexts in the natural way.

We require the following properties:

- **Correctness:** For all $\lambda, d \in \mathbb{N}$ and all messages $\text{msg} \in X_\lambda$, it holds that:

$$\Pr \left[\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, \text{msg})) = \text{msg} : (\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda, 1^d) \right] = 1.$$

- **Evaluation correctness:** For all $\lambda, d \in \mathbb{N}$, all messages $\text{msg} \in X_\lambda$, and all sequences of Boolean circuits $C_1, \dots, C_n$ of total depth at most d , it holds that:

$$\Pr \left[\begin{array}{l} \text{Dec}(\text{sk}, \text{ct}_n) = C_n(\dots C_2(C_1(\text{msg})) \dots) : \\ \begin{array}{l} (\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda, 1^d) \\ \text{ct}_0 \leftarrow \text{Enc}(\text{pk}, \text{msg}) \\ \forall i = 1, \dots, n : \text{ct}_i = \text{Eval}(\text{pk}, C_i, \text{ct}_{i-1}) \end{array} \end{array} \right] = 1.$$

We extend this property to evaluations on multiple ciphertexts in the natural way.

- **Compactness:** There exists a polynomial $\text{poly}(\cdot)$ such that for all $\lambda, d \in \mathbb{N}$, all (sk, pk) in the support of $\text{KeyGen}(1^\lambda, 1^d)$, all messages $\text{msg} \in X_\lambda$, all ciphertexts ct in the support of $\text{Enc}(\text{pk}, \text{msg})$, and all Boolean circuits C , it holds that

$$|\text{Eval}(\text{pk}, C, \text{ct})| \leq \text{poly}(\lambda, \log d, |C(\text{msg})|).$$

- **CPA-security:** For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, define the CPA-security experiment as follows:
 1. On input the security parameter 1^λ , the adversary $\mathcal{A}$ starts by outputting a depth bound 1^d .
 2. The challenger samples a key pair $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda, 1^d)$ and sends pk to the adversary.
 3. The adversary can now make (arbitrarily many) queries on pairs of messages $(\text{msg}_0, \text{msg}_1)$. On each query, the challenger replies with $\text{Enc}(\text{pk}, \text{msg}_b)$.
 4. After the adversary $\mathcal{A}$ is done making queries, it outputs a guess $b' \in \{0, 1\}$.

We say that Π_{LHE} satisfies CPA security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

Polylogarithmic depth decryption. A long sequence of works (cf. [BV11, BGV12, Bra12, GSW13]) has shown how to construct leveled homomorphic encryption schemes from the plain learning with errors (LWE) assumption. Moreover, in these constructions, the decryption circuit can be computed by a circuit of polylogarithmic depth, which we formally define as follows:

Definition A.2 (Polylogarithmic Depth Decryption). We say that a leveled homomorphic encryption scheme $\Pi_{\text{LHE}} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ supports *polylogarithmic depth decryption* if there exists a polynomial p such that for any $\lambda \in \mathbb{N}$, any depth bound $d \in \mathbb{N}$, and any key pair (sk, pk) in the support of $\text{KeyGen}(1^\lambda, 1^d)$, the depth of the decryption circuit computing $\text{Dec}(\text{sk}, \cdot)$ is at most $p(\lambda, \log d)$.

Theorem A.3 (Leveled Homomorphic Encryption [BV11, BGV12, Bra12, GSW13]). *Assuming the hardness of the plain learning with errors (LWE) problem, there exists a leveled homomorphic encryption scheme Π_{LHE} with polylogarithmic depth decryption.*

A.2 Construction

The hash tree. We fix once and for all a canonical block $x_0 \in \mathcal{X}$. The algorithms below set $h = \lceil \log k \rceil$ and Hash pads its input vector to the 2^h leaves of the tree with copies of x_0 , so that Definition 4.1 is met for every $k \in \mathbb{N}$ and not only for powers of two. We hash the blocks along a complete binary tree T with 2^h leaves, whose leaves are labeled by $1, \dots, 2^h$ in left-to-right order and whose root we denote by root . When k is not a power of two we have $2^h > k$; we therefore index leaves by $[2^h] \supseteq [k]$ throughout and reserve the symbol k for the number of blocks passed to Setup. The *height* of a node is its distance from the leaves, so the leaves have height 0 and the root root has height h . A node of height ℓ covers a contiguous range of 2^ℓ leaves. For an index $i \in [2^h]$ and a height $\ell \in [h]$, we write $v_\ell(i)$ for the node of height ℓ covering i , so that $i, v_1(i), \dots, v_h(i)$ is the path from the leaf i to the root. We navigate this path by the binary representation of $i - 1$: for any positive integer i and any $\ell \geq 1$ we write

$$\text{bit}_\ell(i) := \left\lfloor \frac{i-1}{2^{\ell-1}} \right\rfloor \bmod 2$$

for the ℓ^{th} least significant bit of $i - 1$ (so $\text{bit}_\ell(i) = 0$ whenever $2^{\ell-1} \geq i$). For $i \in [2^h]$ and $\ell \in [h]$, the leaf i lies in the left subtree of $v_\ell(i)$ if $\text{bit}_\ell(i) = 0$, and in its right subtree otherwise; equivalently, $\text{bit}_h(i), \dots, \text{bit}_1(i)$ is the sequence of left–right turns on the path from the root down to the leaf i .

Construction A.4 (Dynamic Somewhere-Extractable Hash). The construction uses a single ingredient, a leveled homomorphic encryption scheme $\Pi_{\text{LHE}} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ as per Definition A.1. We require that its message space contains the blocks of $\mathcal{X}$, the strings $\{0, 1\}^{s_f}$ (which encode the descriptions of the functions $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f), the bits $\{0, 1\}$, and the secret keys of Π_{LHE} itself as well as pairs of such keys. Following the blueprint of Hubáček and Wichs [HW15], we use one key pair for every height of the tree, encrypt the secret key of each height under the public key of the height above it, and encrypt the target index one bit per height, so that the ciphertext supplied at height ℓ already contains the single bit that decides which child to propagate there. The difference is that in our construction these bits are not fixed at setup: they are computed homomorphically at key-generation time from the encryption of the function f and the tag τ . The digest is a ciphertext under the key of the root. We write $\text{Const}[x]$ for the circuit that ignores its input and outputs the block x .

Constants: A tag $\tau \in \mathcal{T}$.

Inputs: A string $\hat{f} \in \{0, 1\}^{s_f}$.

1. If $\hat{f}$ is the description of a circuit of depth at most s_d computing a function $g: \mathcal{T} \rightarrow [2^\lambda]$, output $i = g(\tau)$; otherwise output $i = 1$.

Figure 8: Circuit $\text{Index}[\tau]$.

Constants: A ciphertext ct_τ .

Inputs: A secret key sk' .

1. Set $i = \text{LHE.Dec}(\text{sk}', \text{ct}_\tau)$ and output i if $i \in [2^\lambda]$, and 1 otherwise.

Figure 9: Circuit $\text{ReEnc}[\text{ct}_\tau]$.

Constants: A height $\ell \in [h]$ and ciphertexts ct_τ , c_L and c_R .

Inputs: A secret key sk' .

1. Set $i = \text{LHE.Dec}(\text{sk}', \text{ct}_\tau)$ if this value lies in $[2^\lambda]$, and $i = 1$ otherwise.
2. Set $x = \text{LHE.Dec}(\text{sk}', c_L)$ if $\text{bit}_\ell(i) = 0$, and $x = \text{LHE.Dec}(\text{sk}', c_R)$ otherwise.
3. Output x if $x \in \mathcal{X}$, and the canonical block x_0 otherwise.

Figure 10: Circuit $\text{Sel}[\ell, \text{ct}_\tau, c_L, c_R]$.

The algorithms of the scheme are defined as follows:

- **Setup**($1^\lambda, k, 1^{s_f}, 1^{s_d}$): On input the security parameter $\lambda \in \mathbb{N}$, the number of blocks $k \in \mathbb{N}$, a description size bound $s_f \in \mathbb{N}$, and a depth bound $s_d \in \mathbb{N}$, the setup algorithm proceeds as follows:
 1. Set $h = \lceil \log k \rceil$ and sample, for each height $\ell \in [0, h]$, a key pair $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$, where $d := d(\lambda, k, s_f, s_d)$ is a bound on the depth of the circuits of Figure 8, Figure 9 and Figure 10.
 2. Encrypt the all-zeros string, $\text{ct}_f \leftarrow \text{LHE.Enc}(\text{pk}_0, 0^{s_f})$.
 3. For each height $\ell \in [h]$, encrypt $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, 0^n)$, where $n = n(\lambda, d)$ is the length of the secret keys of $\text{LHE.KeyGen}(1^\lambda, 1^d)$, which we may assume to be the same for all of them by padding.
 4. Output $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$.
- **SetupBinding**($1^\lambda, k, 1^{s_f}, 1^{s_d}, f$): On input the security parameter $\lambda \in \mathbb{N}$, the number of blocks $k \in \mathbb{N}$, a description size bound $s_f \in \mathbb{N}$, a depth bound $s_d \in \mathbb{N}$, and (the description of) a function $f: \mathcal{T} \rightarrow [2^\lambda]$, the binding setup algorithm proceeds as follows:
 1. Set $h = \lceil \log k \rceil$ and sample, for each height $\ell \in [0, h]$, a key pair $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$, with the same depth bound $d = d(\lambda, k, s_f, s_d)$ as in Setup.
 2. Encrypt the function, $\text{ct}_f \leftarrow \text{LHE.Enc}(\text{pk}_0, f)$, where the description of f is padded to exactly s_f bits.
 3. For each height $\ell \in [h]$, encrypt $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, \text{sk}_{\ell-1})$.
 4. Output $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ and $\text{td} = \text{sk}_h$.
- **Gen**(pp, τ): On input the public parameters $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ and a tag $\tau \in \mathcal{T}$, the key-generation algorithm sets

$$\text{ct}_{\tau,0} := \text{LHE.Eval}(\text{pk}_0, \text{Index}[\tau], \text{ct}_f) \quad \text{and} \quad \text{ct}_{\tau,\ell} := \text{LHE.Eval}(\text{pk}_\ell, \text{ReEnc}[\text{ct}_{\tau,\ell-1}], \text{ct}_\ell) \quad \text{for } \ell \in [h-1],$$

and outputs $\text{hk} = ((\text{ct}_{\tau,\ell})_{\ell \in [0, h-1]}, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$. The algorithm is deterministic because LHE.Eval is.

- **Hash**($\text{hk}, (x_1, \dots, x_k)$): On input a hash key $\text{hk} = ((\text{ct}_{\tau,\ell})_{\ell \in [0, h-1]}, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ and a vector of blocks $(x_1, \dots, x_k) \in \mathcal{X}^k$, the hash algorithm proceeds as follows:

1. For each leaf $i \in [2^h]$, set $c_i = \text{LHE.Eval}(\text{pk}_0, \text{Const}[x_i], \text{ct}_f)$, where $x_i := x_0$ for $i > k$; that is, the vector is padded to the full tree with copies of the canonical block.
2. For each internal node v of height $\ell \in [h]$, with children L and R , set

$$c_v = \text{LHE.Eval}(\text{pk}_\ell, \text{Sel}[\ell, \text{ct}_{\tau, \ell-1}, c_L, c_R], \text{ct}_\ell),$$

proceeding from the leaves upwards, where Sel is the circuit of [Figure 10](#).

3. Output the digest $\text{dig} = c_{\text{root}}$ and, for each leaf $i \in [k]$, the opening $\sigma_i = (c_{v'_1(i)}, \dots, c_{v'_h(i)})$, where $v'_\ell(i)$ denotes the sibling of $v_{\ell-1}(i)$ (writing $v_0(i) := i$); that is, the opening consists of the ciphertexts of the siblings of the nodes on the path from i to the root, ordered by increasing height.
- **Verify**($\text{hk}, \text{dig}, i, x, \sigma$): On input a hash key $\text{hk} = ((\text{ct}_{\tau, \ell})_{\ell \in [0, h-1]}, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$, which determines the height h of the tree and hence its number of leaves 2^h , a digest dig , an index $i \in \mathbb{N}$, a block $x \in \mathcal{X}$, and an opening $\sigma = (s_1, \dots, s_h)$, the verification algorithm proceeds as follows:

1. Output 0 if the opening is not of this format or if $i \notin [2^h]$.
2. Set $c^{(0)} = \text{LHE.Eval}(\text{pk}_0, \text{Const}[x], \text{ct}_f)$.
3. For each height $\ell = 1, \dots, h$, set

$$c^{(\ell)} = \begin{cases} \text{LHE.Eval}(\text{pk}_\ell, \text{Sel}[\ell, \text{ct}_{\tau, \ell-1}, c^{(\ell-1)}, s_\ell], \text{ct}_\ell) & \text{bit}_\ell(i) = 0 \\ \text{LHE.Eval}(\text{pk}_\ell, \text{Sel}[\ell, \text{ct}_{\tau, \ell-1}, s_\ell, c^{(\ell-1)}], \text{ct}_\ell) & \text{bit}_\ell(i) = 1 \end{cases}$$

so that $c^{(\ell-1)}$ is placed on the side on which the leaf i lies.

4. Output 1 if and only if $c^{(h)} = \text{dig}$.
- **Extract**(td, dig): On input the trapdoor $\text{td} = \text{sk}_h$ and a digest dig , the extraction algorithm computes $x = \text{LHE.Dec}(\text{sk}_h, \text{dig})$, and outputs x if $x \in \mathcal{X}$, and the canonical block x_0 otherwise.

Observe that no secret key is ever encrypted under its own public key: in binding mode the keys $\text{sk}_0, \dots, \text{sk}_{h-1}$ are encrypted only under $\text{pk}_1, \dots, \text{pk}_h$, and sk_h is not encrypted at all, while in normal mode no secret key is encrypted at all. The construction therefore does not rely on any form of circular security.

We now show that [Construction A.4](#) satisfies the properties of [Definition 4.1](#).

Lemma A.5 (Correctness). *Suppose that Π_{LHE} is correct and satisfies evaluation correctness. Then [Construction A.4](#) is correct.*

Proof. For correctness, fix $k \in \mathbb{N}$, a tag τ , a vector of blocks and a leaf $i \in [k]$, and let $\{c_v\}$ be the ciphertexts computed by Hash. The verifier does not reject in its first step: the opening is of the right format and $i \in [k] \subseteq [2^h]$. On input the opening σ_i and the block x_i , the verifier recomputes exactly the ciphertexts of the nodes on the path from i to the root: its $c^{(0)}$ is the leaf ciphertext c_i , and its $c^{(\ell)}$ is obtained by hardwiring $c^{(\ell-1)}$ and the ℓ^{th} entry of σ_i into Sel on the correct sides, with the same ciphertext $\text{ct}_{\tau, \ell-1}$ that Hash used for every node of height ℓ . Indeed, $\text{bit}_\ell(i) = 0$ precisely when the leaf i lies in the left subtree of $v_\ell(i)$, so the verifier places $c^{(\ell-1)}$ as the child of $v_\ell(i)$ that contains i and $s_\ell = c_{v'_\ell(i)}$ as its sibling, which is how Hash combined them. Since LHE.Eval is deterministic, these are the same ciphertexts, so $c^{(h)} = c_{\text{root}}$ and the verifier accepts. $\square$

Lemma A.6 (Succinctness). *Suppose that Π_{LHE} is compact and supports polylogarithmic depth decryption. Then [Construction A.4](#) is succinct.*

Proof. We first bound the depth d that Setup hands to LHE.KeyGen . The circuit of [Figure 8](#) evaluates a universal circuit for the class of circuits of description size at most s_f and depth at most s_d , which has depth $O(s_d \cdot \log s_f)$. The circuits of [Figure 9](#) and [Figure 10](#) perform at most two decryptions under keys in the support of $\text{LHE.KeyGen}(1^\lambda, 1^d)$, one extraction of a bit of the index, which costs one subtraction on at most λ bits, of depth $O(\log \lambda)$, one selection on a single bit, and the membership tests of their final steps, which are the tests that Extract performs and hence

have depth $\text{poly}(\lambda)$. So by [Definition A.2](#) their depth is at most $p(\lambda, \log d) + \text{poly}(\lambda)$, where p is the polynomial of that definition. It therefore suffices to take any d with

$$d \geq O(s_d \cdot \log s_f) + p(\lambda, \log d) + \text{poly}(\lambda),$$

and since the right-hand side grows only polylogarithmically in d , this is satisfied by some $d \leq \text{poly}(\lambda, \log k, s_d, \log s_f)$. Note that d depends on s_f only logarithmically, which is what makes the trapdoor bound below small enough to be composed along a tree. Moreover, if p is efficiently computable, then so is d .

The hash key consists of the $h = \lceil \log k \rceil$ ciphertexts $\text{ct}_{\tau,0}, \dots, \text{ct}_{\tau,h-1}$ together with $h+1$ public keys and h freshly encrypted ciphertexts. Since $\text{LHE.KeyGen}(1^\lambda, 1^d)$ and LHE.Enc run in polynomial time, each public key and each ct_ℓ has size $\text{poly}(\lambda, d)$, and each $\text{ct}_{\tau,\ell}$ is an evaluated ciphertext whose plaintext is an index in $[2^\lambda]$, so compactness gives $|\text{ct}_{\tau,\ell}| \leq \text{poly}(\lambda, \log d)$ and hence $\sum_{\ell \in [0, h-1]} |\text{ct}_{\tau,\ell}| \leq \lceil \log k \rceil \cdot \text{poly}(\lambda, \log d)$. As $d \leq \text{poly}(\lambda, \log k, s_d, \log s_f)$, we get $|hk| \leq \text{poly}(\lambda, \log k, s_d, \log s_f)$. A digest is a single evaluated ciphertext whose plaintext is a block, which has a description of size $\text{poly}(\lambda)$, so compactness gives $|\text{dig}| \leq \text{poly}(\lambda, \log d)$; an opening consists of $h = \lceil \log k \rceil$ such ciphertexts, so $|\sigma_i| \leq \text{poly}(\lambda, \log k, s_d, \log s_f)$. Here we used that the circuits of [Figure 9](#) and [Figure 10](#) output an element of $[2^\lambda]$ and a block of $\mathcal{X}$ respectively, by their final steps, so that $|C(\text{msg})| \leq \text{poly}(\lambda)$ in the compactness bound of [Definition A.1](#) whatever the plaintexts of the ciphertexts they are evaluated on, and in particular in normal mode, where every ct_ℓ encrypts zeros and the decryptions these circuits perform are under a key that is not the one the ciphertexts were produced under. Without these final steps the plaintext could be any element of the message space, which by [Construction A.4](#) contains $\{0, 1\}^{s_f}$, and compactness would only give a bound polynomial in s_f rather than in $\log s_f$. Again, the bound here is also efficiently computable.

For the succinct trapdoor, $\text{td} = \text{sk}_h$ and $\text{LHE.KeyGen}(1^\lambda, 1^d)$ runs in polynomial time, so $|\text{td}| \leq \text{poly}(\lambda, d) \leq \text{poly}(\lambda, \log k, s_d, \log s_f)$. For most LHE constructions, the length of the secret key is efficiently computable. The algorithm $\text{Extract}(\text{td}, \cdot)$ runs $\text{LHE.Dec}(\text{sk}_h, \cdot)$ and tests membership in $\mathcal{X}$, so its circuit has size polynomial in $|\text{td}|$ and $|\text{dig}|$, and hence at most $\text{poly}(\lambda, \log k, s_d, \log s_f)$, and by [Definition A.2](#) its depth is at most $p(\lambda, \log d) = \text{poly}(\lambda, \log k, \log s_d, \log s_f)$. $\square$

Lemma A.7 (Mode Indistinguishability). *Assume that Π_{LHE} is CPA-secure. Then [Construction A.4](#) satisfies mode indistinguishability.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game of [Definition 4.1](#) against [Construction A.4](#). The advantage of $\mathcal{A}$ is the average, over its first message, of its advantage conditioned on that message, so it suffices to bound the difference for the first message $k, 1^{s_f}, 1^{s_d}, f$ that maximizes it. Fix such a message and write $h := \lceil \log k \rceil$ and $d := d(\lambda, k, s_f, s_d)$ for the depth bound of [Construction A.4](#); both are then fixed functions of λ , and $h \leq \text{poly}(\lambda)$ since $\mathcal{A}$ is efficient. We write $n = n(\lambda, d)$ for the common length of the secret keys, as in [Construction A.4](#). All the experiments below differ only in how the challenger samples the public parameters $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ that it sends to $\mathcal{A}$; in each of them the output of the experiment is the guess b' that $\mathcal{A}$ outputs, and the trapdoor is never given to $\mathcal{A}$. The goal is to prove that the adversary cannot distinguish between the two following experiments except with negligible probability:

- $\text{Real}^{(0)}$: the mode indistinguishability game with $b = 0$, in which the public parameters are sampled in normal mode. Formally:
 1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^{s_f}, 1^{s_d}$ and the description of a function $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d .
 2. The challenger runs $\text{Setup}(1^\lambda, k, 1^{s_f}, 1^{s_d})$. That is:
 - Sample a key pair $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$ for every height $\ell \in [0, h]$.
 - Encrypt the all-zeros string, $\text{ct}_f \leftarrow \text{LHE.Enc}(\text{pk}_0, 0^{s_f})$.
 - Encrypt $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, 0^n)$ for every height $\ell \in [h]$.
The challenger sends $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.

- $\text{Real}^{(1)}$: the mode indistinguishability game with $b = 1$, in which the public parameters are sampled in binding mode. Formally:

1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^{s_f}, 1^{s_d}$ and the description of a function $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d .
2. The challenger runs $\text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f)$. That is:
 - Sample a key pair $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$ for every height $\ell \in [0, h]$.
 - Encrypt the function, $\text{ct}_f \leftarrow \text{LHE.Enc}(\text{pk}_0, f)$, where the description of f is padded to exactly s_f bits.
 - Encrypt $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, \text{sk}_{\ell-1})$ for every height $\ell \in [h]$.

The challenger sends $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ to $\mathcal{A}$ and keeps the trapdoor $\text{td} = \text{sk}_h$.
3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.

To do so, we define the following sequence of hybrids, one for each $j \in [0, h]$:

- Hyb_j : the same as $\text{Real}^{(1)}$, except that only the ciphertexts of the heights $\ell \leq j$ encrypt secret keys, whereas those of the heights $\ell > j$ encrypt zeros. Formally:

2. The challenger samples the key pairs $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$ for every height $\ell \in [0, h]$ and computes $\text{ct}_f \leftarrow \text{LHE.Enc}(\text{pk}_0, f)$ as in $\text{Real}^{(1)}$, and then encrypts:
 - $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, \text{sk}_{\ell-1})$ for every height $\ell \in [j]$.
 - $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, 0^n)$ for every height $\ell \in [j+1, h]$.

The rest of the experiment remains unchanged.

In particular, Hyb_h is identical to $\text{Real}^{(1)}$, and in Hyb_0 every ciphertext ct_ℓ encrypts zeros, exactly as in $\text{Real}^{(0)}$, so Hyb_0 and $\text{Real}^{(0)}$ differ only in the plaintext of ct_f . The chain of experiments is

$$\text{Real}^{(1)} = \text{Hyb}_h, \text{Hyb}_{h-1}, \dots, \text{Hyb}_1, \text{Hyb}_0, \text{Real}^{(0)},$$

where a step from Hyb_j to Hyb_{j-1} replaces the encryption of sk_{j-1} under pk_j by an encryption of zeros, erasing the secret keys from the top of the tree downwards, and the last step switches the plaintext of ct_f . We bound the two kinds of steps separately. In the first claim we bound the *signed* sum over the heights rather than each height separately: this is what a single reduction that samples the height can bound, whereas bounding each height separately would require one negligible function per height, and the number of heights grows with λ .

Claim A.8. Assume that Π_{LHE} is CPA-secure. Then there exists a negligible function negl^{sk} such that

$$|\Pr[\text{Hyb}_h(\mathcal{A}) = 1] - \Pr[\text{Hyb}_0(\mathcal{A}) = 1]| \leq h \cdot \text{negl}^{\text{sk}}(\lambda).$$

Proof. Consider the following efficient adversary $\mathcal{B}$ for the CPA-security game of Π_{LHE} (Definition A.1), which takes a height $j \in [h]$ as an input. We write β for the bit of the CPA-security challenger.

1. Send the depth bound 1^d to the challenger, and let pk be the public key it returns. Set $\text{pk}_j = \text{pk}$.
2. Sample $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$ for every height $\ell \in [0, h] \setminus \{j\}$.
3. Query the challenger on the message pair $(0^n, \text{sk}_{j-1})$, and let ct_j be the ciphertext it returns. The key sk_{j-1} is available because $j \geq 1$, so the height $j-1$ lies in $[0, h] \setminus \{j\}$.
4. Set $\text{ct}_f \leftarrow \text{LHE.Enc}(\text{pk}_0, f)$, with the description of f padded to exactly s_f bits.
5. For every height $\ell \in [h] \setminus \{j\}$, set $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, \text{sk}_{\ell-1})$ if $\ell < j$ and $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, 0^n)$ if $\ell > j$.
6. Send $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

The adversary $\mathcal{B}$ is efficient: it runs $\mathcal{A}$ once, samples h further key pairs and makes a single query, on a pair of messages of equal length n .

The simulation is perfect in both modes. The public keys are sampled as in both experiments, the ciphertext ct_f encrypts f as in both experiments, every height $\ell < j$ encrypts the secret key of the height below it and every height $\ell > j$ encrypts zeros, in both cases as the challengers of Hyb_{j-1} and of Hyb_j do. The only difference is the ciphertext of height j : if the CPA-security challenger is in mode $\beta = 0$ then ct_j is distributed as $\text{LHE.Enc}(\text{pk}_j, 0^n)$ and $\mathcal{B}(j)$ perfectly simulates Hyb_{j-1} , and if it is in mode $\beta = 1$ then ct_j is distributed as $\text{LHE.Enc}(\text{pk}_j, \text{sk}_{j-1})$ and $\mathcal{B}(j)$ perfectly simulates Hyb_j . Note that $\mathcal{B}$ never uses sk_j , which the challenger does not give it: the only ciphertext under pk_j is ct_j , which the challenger produces, and sk_j is the plaintext of ct_{j+1} only when $j+1 \leq h$, where both experiments encrypt zeros instead. It is because the keys are erased from the top downwards that this is so: at the step that erases ct_j , every height above j has already been erased. Consequently, for every height j ,

$$\Pr[\mathcal{B}(j) = 1 : \beta = 1] - \Pr[\mathcal{B}(j) = 1 : \beta = 0] = \Pr[\text{Hyb}_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{j-1}(\mathcal{A}) = 1].$$

Let $\mathcal{B}^*$ be the adversary that samples $j \xleftarrow{\mathcal{R}} [h]$ and runs $\mathcal{B}(j)$. It is a single efficient adversary, so by CPA security of Π_{LHE} there is a negligible function negl^{sk} with

$$|\Pr[\mathcal{B}^* = 1 : \beta = 1] - \Pr[\mathcal{B}^* = 1 : \beta = 0]| \leq \text{negl}^{\text{sk}}(\lambda).$$

Since j is sampled uniformly and independently of the challenger's bit, the left-hand side equals

$$\left| \frac{1}{h} \sum_{j \in [h]} \left(\Pr[\mathcal{B}(j) = 1 : \beta = 1] - \Pr[\mathcal{B}(j) = 1 : \beta = 0] \right) \right| = \frac{1}{h} \cdot |\Pr[\text{Hyb}_h(\mathcal{A}) = 1] - \Pr[\text{Hyb}_0(\mathcal{A}) = 1]|,$$

where the sum telescopes, and the claim follows by multiplying by h . $\square$

Claim A.9. Assume that Π_{LHE} is CPA-secure. Then there exists a negligible function negl^f such that

$$|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Real}^{(0)}(\mathcal{A}) = 1]| \leq \text{negl}^f(\lambda).$$

Proof. Consider the following efficient adversary $\mathcal{C}$ for the CPA-security game of Π_{LHE} (Definition A.1), where again β denotes the bit of the challenger.

1. Send the depth bound 1^d to the challenger, and let pk be the public key it returns. Set $\text{pk}_0 = \text{pk}$.
2. Sample $(\text{sk}_\ell, \text{pk}_\ell) \leftarrow \text{LHE.KeyGen}(1^\lambda, 1^d)$ for every height $\ell \in [h]$.
3. Query the challenger on the message pair $(0^{s_f}, f)$, with the description of f padded to exactly s_f bits, and let ct_f be the ciphertext it returns.
4. For every height $\ell \in [h]$, set $\text{ct}_\ell \leftarrow \text{LHE.Enc}(\text{pk}_\ell, 0^n)$.
5. Send $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

The adversary $\mathcal{C}$ is efficient, and its single query is on a pair of messages of equal length s_f .

The simulation is perfect in both modes. The public keys are sampled as in both experiments and every ciphertext ct_ℓ encrypts zeros, as in Hyb_0 and in $\text{Real}^{(0)}$. The only difference is the plaintext of ct_f : if $\beta = 0$ then ct_f is distributed as $\text{LHE.Enc}(\text{pk}_0, 0^{s_f})$ and $\mathcal{C}$ perfectly simulates $\text{Real}^{(0)}$, and if $\beta = 1$ then ct_f is distributed as $\text{LHE.Enc}(\text{pk}_0, f)$ and $\mathcal{C}$ perfectly simulates Hyb_0 . Note that $\mathcal{C}$ never uses sk_0 , which the challenger does not give it: in both experiments every ct_ℓ encrypts zeros, so sk_0 is the plaintext of no ciphertext. Hence

$$\Pr[\mathcal{C} = 1 : \beta = 1] - \Pr[\mathcal{C} = 1 : \beta = 0] = \Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Real}^{(0)}(\mathcal{A}) = 1],$$

and the claim follows from CPA security of Π_{LHE} . $\square$

Combining [Claim A.8](#) with [Claim A.9](#) and the triangle inequality along the chain,

$$\left| \Pr[\text{Real}^{(1)}(\mathcal{A}) = 1] - \Pr[\text{Real}^{(0)}(\mathcal{A}) = 1] \right| \leq h \cdot \text{negl}^{\text{sk}}(\lambda) + \text{negl}^f(\lambda),$$

which is negligible since $h \leq \text{poly}(\lambda)$. Since $\text{Real}^{(0)}$ and $\text{Real}^{(1)}$ are the mode indistinguishability game with $b = 0$ and with $b = 1$, respectively, this is exactly mode indistinguishability of [Construction A.4](#). $\square$

Lemma A.10 (Somewhere Extraction). *Suppose that Π_{LHE} is correct and satisfies evaluation correctness. Then [Construction A.4](#) satisfies somewhere extraction.*

Proof. Fix $\lambda, s_f, s_d, k \in \mathbb{N}$, a function $f: \mathcal{T} \rightarrow [2^\lambda]$ of description size at most s_f and depth at most s_d , a tag $\tau \in \mathcal{T}$ and a pair (pp, td) in the support of $\text{SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f)$, and write $h = \lceil \log k \rceil$. Write $\text{pp} = (\text{ct}_f, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$, $\text{td} = \text{sk}_h$ and $\text{hk} = \text{Gen}(\text{pp}, \tau) = ((\text{ct}_{\tau, \ell})_{\ell \in [0, h-1]}, (\text{pk}_\ell)_{\ell \in [0, h]}, (\text{ct}_\ell)_{\ell \in [h]})$, and denote $i^* := f(\tau)$. The ciphertext ct_f is a fresh encryption of the $(s_f$ -bit padded) description of f under pk_0 , and $\text{ct}_{\tau, 0}$ is obtained from it by a single evaluation of $\text{Index}[\tau]$, a circuit of depth at most d , so evaluation correctness gives $\text{LHE.Dec}(\text{sk}_0, \text{ct}_{\tau, 0}) = \text{Index}[\tau](f) = i^*$. Inductively, if $\text{LHE.Dec}(\text{sk}_{\ell-1}, \text{ct}_{\tau, \ell-1}) = i^*$ then $\text{ct}_{\tau, \ell} = \text{LHE.Eval}(\text{pk}_\ell, \text{ReEnc}[\text{ct}_{\tau, \ell-1}], \text{ct}_\ell)$ and ct_ℓ is a fresh encryption of $\text{sk}_{\ell-1}$ under pk_ℓ , so evaluation correctness gives $\text{LHE.Dec}(\text{sk}_\ell, \text{ct}_{\tau, \ell}) = \text{ReEnc}[\text{ct}_{\tau, \ell-1}](\text{sk}_{\ell-1}) = i^*$, where the final step of [Figure 9](#) leaves the value unchanged because $i^* \in [2^\lambda]$; hence

$$\text{LHE.Dec}(\text{sk}_\ell, \text{ct}_{\tau, \ell}) = i^* \quad \text{for every } \ell \in [0, h-1].$$

Now fix any digest dig , any block $x \in \mathcal{X}$ and any opening σ , and assume that $\text{Verify}(\text{hk}, \text{dig}, i^*, x, \sigma) = 1$. Since the verifier accepts, its first step did not reject: the opening parses as $\sigma = (s_1, \dots, s_h)$ and $i^* \in [2^h]$. Let $c^{(0)}, \dots, c^{(h)}$ be the ciphertexts that the verifier recomputes. We prove by induction on $\ell \in [0, h]$ that $\text{LHE.Dec}(\text{sk}_\ell, c^{(\ell)}) = x$.

For $\ell = 0$, the verifier sets $c^{(0)} = \text{LHE.Eval}(\text{pk}_0, \text{Const}[x], \text{ct}_f)$, where ct_f is a fresh encryption under pk_0 , so evaluation correctness gives $\text{LHE.Dec}(\text{sk}_0, c^{(0)}) = x$, since $\text{Const}[x]$ ignores its input.

For the inductive step, let $\ell \in [h]$ and assume that $\text{LHE.Dec}(\text{sk}_{\ell-1}, c^{(\ell-1)}) = x$. The verifier sets $c^{(\ell)} = \text{LHE.Eval}(\text{pk}_\ell, \text{Sel}[\ell, \text{ct}_{\tau, \ell-1}, \cdot, \cdot], \text{ct}_\ell)$, where $c^{(\ell-1)}$ is hardwired as the left child if $\text{bit}_\ell(i^*) = 0$ and as the right child otherwise. Here ct_ℓ is a fresh encryption of $\text{sk}_{\ell-1}$ under pk_ℓ , and Sel has depth at most d , so evaluation correctness gives

$$\text{LHE.Dec}(\text{sk}_\ell, c^{(\ell)}) = \text{Sel}[\ell, \text{ct}_{\tau, \ell-1}, \cdot, \cdot](\text{sk}_{\ell-1}).$$

By the specification of [Figure 10](#), the circuit first computes $\text{LHE.Dec}(\text{sk}_{\ell-1}, \text{ct}_{\tau, \ell-1}) = i^*$ by the display above, which lies in $[2^\lambda]$ and is therefore not replaced by its first step, then the bit $\text{bit}_\ell(i^*)$, and then takes the decryption under $\text{sk}_{\ell-1}$ of the left child if that bit is 0 and of the right child otherwise. This is the same side on which the verifier hardwired $c^{(\ell-1)}$, since the verifier selects that side by the very same bit $\text{bit}_\ell(i^*)$. The circuit therefore takes $\text{LHE.Dec}(\text{sk}_{\ell-1}, c^{(\ell-1)})$, which equals x by the induction hypothesis, and outputs it, since $x \in \mathcal{X}$ and its last step therefore does not fall back on the canonical block. Note that this holds regardless of the sibling ciphertexts in σ : the selection is made by the encrypted bit and not by the ciphertexts being combined.

Since the verifier accepts, we have $\text{dig} = c^{(h)}$, and therefore $\text{LHE.Dec}(\text{sk}_h, \text{dig}) = x \in \mathcal{X}$, so Extract does not fall back on the canonical block and

$$\text{Extract}(\text{td}, \text{dig}) = \text{LHE.Dec}(\text{sk}_h, \text{dig}) = \text{LHE.Dec}(\text{sk}_h, c^{(h)}) = x,$$

as required. The argument used no probability, so the property holds perfectly, and it applies to every digest. $\square$

B Statistical Functional BARGs

In this section, we build a statistical functional BARG as defined in [Section 4.2](#). The construction follows the BARG construction of Choudhuri [\[CJJ21b\]](#) closely, except we use a dynamic somewhere-extractable hash ([Definition 4.1](#)) instead of a plain somewhere-extractable hash [\[HW15\]](#), and we use a somewhere-statistically correlation intractable hash [\[CCH⁺19\]](#) instead of a (computationally) correlation intractable hash [\[CGH98\]](#).

B.1 Building Blocks

The following definition are copied mostly verbatim from a prior work [NWW26, Appendix A].

PCP with fast online verification. We recall the definition of probabilistically checkable proofs (PCPs) with a *fast online verification property* as written in [CJJ21b], and implicitly considered by Bitansky et al. [BCI⁺13]. At a high level, such a property requires that for any PCP for the circuit satisfiability language

$$\text{C-SAT} = \{x \mid \exists w : C(x, w) = 1\},$$

the verification algorithm can be split into two parts: (i) a query algorithm *Query* which generates the PCP queries that depend on C but are independent of x ; and (ii) an online verification algorithm *Verify*, which depends on x but whose running time grows only polylogarithmically in $|C|$ and polynomially in n .

Definition B.1 (PCP with Fast Online Verification [BCI⁺13, CJJ21b]). For any Boolean circuit $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$, a PCP with fast online verification for C-SAT is a tuple of polynomial-time algorithms (*Prove*, *Query*, *Verify*) with the following syntax:

- *Prove*($1^\lambda, C, x, w$) $\rightarrow \pi$: The prover algorithm takes as input a security parameter λ , the circuit C , an instance x , and its witness w , and outputs a PCP proof $\pi \in \{0, 1\}^*$.
- *Query*($1^\lambda, C$) $\rightarrow (Q, \text{st})$: On input the security parameter λ and the circuit C , the randomized query algorithm outputs a subset of indices $Q \subseteq [| \pi |]$ and a state st .
- *Verify*($x, \text{st}, \hat{\pi}$) $\rightarrow b$: On input an instance x , a state st , and a binary string $\hat{\pi} \in \{0, 1\}^{|Q|}$ (thought of as the restriction of a proof to the queried positions), the online verification algorithm deterministically decides to accept (outputs 1) or reject (outputs 0).

Furthermore, we require the following properties of the PCP:

- **Completeness:** For any circuit C , any instance $x \in \text{C-SAT}$, and any witness w for x , we have:

$$\Pr \left[\text{Verify}(x, \text{st}, \pi|_Q) = 1 \mid \begin{array}{l} \pi \leftarrow \text{Prove}(1^\lambda, C, x, w) \\ (Q, \text{st}) \xleftarrow{\mathcal{R}} \text{Query}(1^\lambda, C) \end{array} \right] = 1.$$

- **Polynomial proof size:** The size of the proof π is bounded by $\text{poly}(\lambda, |C|)$. For a circuit size bound $s \in \mathbb{N}$, we write $\ell := \ell(\lambda, s)$ for this bound evaluated at $|C| = s$, and we assume without loss of generality that the prover pads its output to exactly $\ell(\lambda, s)$ bits, so that all circuits of size at most s have proofs of the same length ℓ .
- **Small query complexity:** The size of the set Q is bounded by $\text{poly}(\lambda, \log |C|)$.
- **Succinct verification:** The state st can be represented in $\text{poly}(\lambda, n, \log |C|)$ bits, and the online verification algorithm *Verify* runs in time $\text{poly}(\lambda, n, \log |C|)$. The query algorithm *Query* runs in time $\text{poly}(\lambda, |C|)$.
- **ρ -Proof of knowledge:** There exists a deterministic polynomial-time extractor algorithm E . We define the following experiment for any unbounded adversary $\mathcal{A}$:
 1. On input a security parameter 1^λ , the adversary $\mathcal{A}$ sends a circuit C , an instance x , and a proof $\pi^* \in \{0, 1\}^*$ to the challenger.
 2. The challenger computes the exact success probability of the proof:

$$\varepsilon = \Pr_{(Q, \text{st}) \leftarrow \mathcal{R}_{\text{Query}(1^\lambda, C)}} [\text{Verify}(x, \text{st}, \pi^*|_Q) = 1].$$

3. If $\varepsilon \leq \rho(\lambda)$, the experiment aborts and outputs 0.

4. The challenger extracts the witness by computing $w \leftarrow E(\pi^*)$.
5. The experiment outputs 1 if $C(x, w) = 0$, and 0 otherwise.

We say that the PCP is a proof of knowledge if for any unbounded adversary, the experiment always outputs 0.

Lemma B.2 (PCP with Fast Online Verification [CJJ21b]). *There exists a PCP with fast online verification for C-SAT that is a $1 - 1/\text{polylog}(|C|)$ -proof of knowledge, where C is the input circuit to Prove.*

Relations and correlation intractability. We recall some definitions from [CGH98, CCH⁺19, HLR21, CJJ21b] and a result that is due to the combination of [HLR21, CJJ21b, HW26].

Definition B.3 (Somewhere Statistically Correlation Intractable Hash Function [CGH98, CCH⁺19]). Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of relations, where each $R \in \mathcal{R}_\lambda$ is a relation over $\{0, 1\}^{\ell_{\text{in}}} \times \{0, 1\}^{\ell_{\text{out}}}$ specified by a description $\langle R \rangle$. A *somewhere statistically correlation intractable* hash function for $\mathcal{R}$ consists of a tuple of efficient algorithms $\Pi_{\text{CI}} = (\text{Gen}, \text{GenAvoid}, \text{Hash})$ with the following syntax:

- $\text{Gen}(1^\lambda, 1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}) \rightarrow \text{hk}$: On input a security parameter λ , a description size bound s , an input length ℓ_{in} , and an output length ℓ_{out} , the generation algorithm outputs a hash key hk .
- $\text{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, d) \rightarrow \text{hk}$: On input a security parameter λ , a description size bound s , an input length ℓ_{in} , an output length ℓ_{out} , and a description d of size at most s , the avoiding generation algorithm outputs a hash key hk .
- $\text{Hash}(\text{hk}, x) \rightarrow y$: On input a hash key hk and a string $x \in \{0, 1\}^{\ell_{\text{in}}}$, the evaluation algorithm outputs a string $y \in \{0, 1\}^{\ell_{\text{out}}}$.

The algorithms should satisfy the two following properties:

- **Statistical correlation intractability:** For any (potentially unbounded) adversary $\mathcal{A}$, we define the following experiment:
 1. On input a security parameter 1^λ , the adversary $\mathcal{A}$ sends $1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}$ as well as a description $\langle R \rangle$ of size at most s , where $R \in \mathcal{R}_\lambda$.
 2. The challenger samples an avoiding hash key $\text{hk} \leftarrow \text{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, \langle R \rangle)$ and sends hk to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs $x \in \{0, 1\}^{\ell_{\text{in}}}$.

We say that Π_{CI} is statistically correlation intractable if for any $\mathcal{A}$ there exists a negligible function $\text{negl}(\lambda)$ such that in the experiment above, we have $\Pr[(x, \text{Hash}(\text{hk}, x)) \in R] \leq \text{negl}(\lambda)$. We say that Π_{CI} is perfectly correlation intractable if $\text{negl}(\lambda) = 0$.

- **Mode indistinguishability:** For any efficient adversary $\mathcal{A}$, we define the following game parameterized by a bit $b \in \{0, 1\}$:
 1. On input a security parameter 1^λ , the adversary $\mathcal{A}$ sends $1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}$, and the description $\langle R \rangle$ of a relation $R \in \mathcal{R}_\lambda$ to the challenger.
 2. The challenger computes a hash key hk as follows:
 - If $b = 0$, the challenger computes $\text{hk} \leftarrow \text{Gen}(1^\lambda, 1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}})$.
 - If $b = 1$, the challenger computes $\text{hk} \leftarrow \text{GenAvoid}(1^\lambda, 1^s, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, \langle R \rangle)$.
The challenger sends hk to $\mathcal{A}$.
 3. The adversary $\mathcal{A}$ outputs a guess b' which is the output of the experiment.

We say that Π_{CI} satisfies mode indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \text{negl}(\lambda).$$

Definition B.4 (Bad Relation for PCP [CJJ21b]). Let Π_{DSEH} be a dynamic somewhere-extractable hash, and let Π_{PCP} be a PCP with fast online verification. For a security parameter $\lambda \in \mathbb{N}$, a witness length $m := m(\lambda) \in \mathbb{N}$, a number of instances $k := k(\lambda) \in \mathbb{N}$, a circuit size bound $s := s(\lambda) \in \mathbb{N}$, with $\ell := \ell(\lambda, s)$ the corresponding PCP proof length, a function $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f and depth at most s_d , and a dynamic somewhere-extractable hash trapdoor td_{DSEH} output by $\text{DSEH.SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_d}, f)$, we define the bad relation $\text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}}$ as follows.

For any circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , any tag $\tau \in \mathcal{T}$, and any sequence of ℓ digests $\text{dig} = (\text{dig}_1, \dots, \text{dig}_\ell)$, let the extracted PCP proof $\pi^* \in \{0, 1\}^\ell$ be defined bit-by-bit as $\pi^*[j] = \text{DSEH.Extract}(\text{td}_{\text{DSEH}}, \text{dig}_j)$ for all $j \in [\ell]$. Let $w = E(\pi^*)$, where E is the deterministic proof of knowledge extractor for the PCP.

The relation $\text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}}$ consists of pairs $((C, \tau, \text{dig}), r)$, where $r \in \{0, 1\}^{\ell_{\text{out}}}$ is used as the randomness of PCP.Query , with ℓ_{out} the number of random bits that $\text{PCP.Query}(1^\lambda, C)$ consumes on circuits of size at most s , such that:

$$C(f(\tau), w) \neq 1 \quad \wedge \quad \text{PCP.Verify}(f(\tau), \text{st}, \pi^*|_Q) = 1$$

where $(Q, \text{st}) = \text{PCP.Query}(1^\lambda, C; r)$. We denote the family of relations by $\text{BAD}_\lambda = \{\text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}}\}$, where the family ranges over all choices of the parameters m, k and s fixed above and over all pairs $(f, \text{td}_{\text{DSEH}})$ as above; the dependence on m, k and s is left implicit in the notation $\text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}}$. Following Definition B.3, we take the description $\langle \text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}} \rangle$ to be the circuit that, on input $((C, \tau, \text{dig}), r)$, decides membership in $\text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}}$ as above, with the function f , the trapdoor td_{DSEH} , the PCP query and verification algorithms, and the proof-of-knowledge extractor E hardcoded. This circuit has size $p_{\text{BAD}}(\lambda, \log k, s, s_f, s_d)$ for a polynomial p_{BAD} ; unlike in the static setting, it depends on the description of f , since it must evaluate f on the tag.

A sequence of works [HLR21, CJJ21b, HW26, NWW26] proves the existence of a somewhere statistically correlation intractable hash for the relation BAD from Definition B.4 for a PCP with negligible proof of knowledge error, assuming either of circular secure FHE, or witness encryption and LWE.

Proposition B.5 (Correlation Intractability for PCP Bad Challenge Function [HLR21, CJJ21b, HW26, NWW26]). *Assume either of the existence of circular-secure FHE, or the existence of witness encryption and the learning with errors assumption with sub-exponential approximation factors. Then there exists a PCP with fast online verification Π_{PCP} that is a $2^{-\Omega(\lambda)}$ -proof of knowledge and a hash scheme Π_{CI} that is somewhere statistically correlation intractable for the family of relations $\text{BAD} = \{\text{BAD}_\lambda\}$ as defined in Definition B.4 for Π_{PCP} .*

The idea behind the proof of Proposition B.5 is that there exists a somewhere statistically correlation intractable hash for *sparse product relations* assuming witness encryption and LWE. The relation BAD that we get by applying parallel repetition to the PCP of Lemma B.2 is a product relation. Moreover, repeating the PCP a polynomial number of times reduces the proof of knowledge error so that the resulting relation is sparse.

B.2 Constructing BARGs

The construction is a direct generalization of the Choudhuri et al. BARG [CJJ21b]: instead of using a (static) somewhere-extractable hash, we use a dynamic somewhere-extractable hash (Definition 4.1). We construct the functional (index) BARG recursively. The base of the recursion is batches of size $k = 1$: in this case, we use a trivial construction with no CRS nor trapdoor, where the prover just sends its witness and the verifier checks the NP relation (Remark B.9). Given a batch size $k > 1$, Construction B.6 builds a BARG for k instances out of a BARG for $k/2$ instances, which we call the BARG of the previous level, so that the scheme for batch size k is obtained by instantiating the construction $\log k$ times on top of the base scheme. Specifically, on input an instance batch $x_1, \dots, x_k$, witnesses $w_1, \dots, w_k$ and a common reference string derived from a tag τ , the prover generates a PCP proof for every statement in the batch of size k , arranges the PCP proofs in rows and commits to each column using the dynamic somewhere-extractable hash key that the common reference string carries. It then applies the correlation-intractable hash to the digests and the tag, in order to derive the randomness for the PCP queries, and uses the openings of the dynamic somewhere-extractable hash as witnesses for the BARG of the previous level with $k/2$ instances, where each instance $i' \in [k/2]$ just states that the openings are valid and make the PCP verifier accept instances $2i', 2i' - 1 \in [k]$. The common reference string of the previous level is derived from the same tag τ . In the trapdoor mode, the main BARG is set up with some function f , and the previous level is set up with the function $f'(\tau) := \lceil f(\tau)/2 \rceil$.

Parameters. We assume throughout that the batch size k is a power of two, so that $k/2$ is an integer. The security parameter λ and the bounds s, s_f are fixed by the call to Setup or SetupBinding that produced pp , and the batch size k and the witness length m are the index range and the input length of the circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$; we leave all of them implicit in the inputs of Gen, Prove, Verify and Extract. Moreover, we assume that $m(\lambda)$ is known ahead of time as a function of λ . Besides these, [Construction B.6](#) consumes the following bounds.

- $\ell := \ell(\lambda, s)$ is the proof length of Π_{PCP} . Recall that proofs are padded so that all circuits of size at most s have proofs of the same length ℓ , thus ℓ is fixed once λ and the circuit size bound s are.
- s_{new} bounds the size of the circuit NewRel of [Figure 11](#), which is the circuit that this level hands to the previous one.
- s'_f bounds the description size of the function that SetupBinding hands to the previous level. That function hardcodes f and additionally computes $\lceil \cdot/2 \rceil$ on a $\lceil \log k \rceil$ -bit integer, so we set $s'_f := s_f + O(\log k)$.
- s_{CI} bounds the size of the description $\langle \text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}} \rangle$ of the relation of [Definition B.4](#) for the parameters m, k, s at hand, over all functions $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f and all trapdoors td_{DSEH} in the support of $\text{DSEH.SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s_f}, f)$. By [Definition B.4](#) we may take $s_{\text{CI}} = p_{\text{BAD}}(\lambda, \log k, s, s_f, s_f)$, where p_{BAD} is the polynomial of that definition.
- ℓ_{in} bounds the length of the correlation-intractable hash input (C, τ, dig) , namely $\ell_{\text{in}} = s + \ell_\tau + \ell \cdot \ell_{\text{dig}}$, where $\ell_\tau = \text{poly}(\lambda)$ bounds the length of a tag in $\mathcal{T}_\lambda$ and ℓ_{dig} bounds the length of a single digest of Π_{DSEH} on k blocks with the bound s_f .
- ℓ_{out} is, as in [Definition B.4](#), the number of random bits that $\text{PCP.Query}(1^\lambda, C)$ consumes on circuits of size at most s .

Construction B.6 (Recursive Index BARG). The construction uses the following ingredients:

- A functional index BARG $\Pi'_{\text{FB}} = (\text{Setup}', \text{SetupBinding}', \text{Gen}', \text{Prove}', \text{Verify}', \text{Extract}')$ for $k/2$ instances, as per [Definition 4.3](#), for the tag space $\mathcal{T}$.
- A PCP with fast online verification $\Pi_{\text{PCP}} = (\text{PCP.Prove}, \text{PCP.Query}, \text{PCP.Verify})$ with proof length ℓ and deterministic extractor E .
- A dynamic somewhere-extractable hash $\Pi_{\text{DSEH}} = (\text{DSEH.Setup}, \text{DSEH.SetupBinding}, \text{DSEH.Gen}, \text{DSEH.Hash}, \text{DSEH.Verify}, \text{DSEH.Extract})$ as per [Definition 4.1](#), for the block space $\{0, 1\}$ of single bits and the tag space $\mathcal{T}$.
- A somewhere statistically correlation intractable hash $\Pi_{\text{CI}} = (\text{CI.Gen}, \text{CI.GenAvoid}, \text{CI.Hash})$ for the relation ensemble BAD from [Definition B.4](#).

The functional index BARG scheme $\Pi_{\text{FB}} = (\text{Setup}, \text{SetupBinding}, \text{Gen}, \text{Prove}, \text{Verify}, \text{Extract})$ for k instances is defined as follows.

- **Setup** $(1^\lambda, k, 1^s, 1^{s_f})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$ (in binary), a circuit size bound $s \in \mathbb{N}$, and a description size bound $s_f \in \mathbb{N}$, the setup algorithm proceeds as follows:
 1. Sample the parameters of the dynamic somewhere-extractable hash: $pp_{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, k, 1^{s_f}, 1^{s_f})$.
 2. Sample the correlation-intractable hash key: $hk_{\text{CI}} \leftarrow \text{CI.Gen}(1^\lambda, 1^{s_{\text{CI}}}, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}})$.
 3. Sample the parameters of the previous level, on $k/2$ instances:

$$pp' \leftarrow \text{Setup}'(1^\lambda, k/2, 1^{s_{\text{new}}}, 1^{s'_f})$$

4. Output $pp = (pp_{\text{DSEH}}, hk_{\text{CI}}, pp')$.

- **SetupBinding**($1^\lambda, k, 1^s, 1^{s_f}, f$): On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$ (in binary), a circuit size bound $s \in \mathbb{N}$, a description size bound $s_f \in \mathbb{N}$, and (the description of) a function $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f , the binding setup algorithm proceeds as follows:

1. Define the function of the previous level to be $f': \mathcal{T} \rightarrow [k/2]$, which on input a tag τ outputs $\lceil f(\tau)/2 \rceil$, and which hardcodes f ; by the choice of s'_f , the function f' has description size at most s'_f .
2. Sample the binding parameters of the dynamic somewhere-extractable hash for f :

$$(\text{pp}_{\text{DSEH}}, \text{td}_{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k, 1^{s_f}, 1^{s'_f}, f)$$

3. Sample the avoiding correlation-intractable hash key: $\text{hk}_{\text{CI}} \leftarrow \text{CI.GenAvoid}(1^\lambda, 1^{s_{\text{CI}}}, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, \text{BAD}_{\lambda, f, \text{td}_{\text{DSEH}}})$, where BAD is the relation of [Definition B.4](#).
4. Sample the binding parameters of the previous level, on $k/2$ instances:

$$(\text{pp}', \text{td}') \leftarrow \text{SetupBinding}'(1^\lambda, k/2, 1^{s_{\text{new}}}, 1^{s'_f}, f')$$

5. Output $\text{pp} = (\text{pp}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{pp}')$, and $\text{td} = \text{td}_{\text{DSEH}}$.

- **Gen**(pp, τ): On input the public parameters $\text{pp} = (\text{pp}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{pp}')$ and a tag $\tau \in \mathcal{T}$, the generation algorithm proceeds as follows:

1. Compute the dynamic somewhere-extractable hash key $\text{hk}_{\text{DSEH}} = \text{DSEH.Gen}(\text{pp}_{\text{DSEH}}, \tau)$.
2. Derive the common reference string of the previous level with respect to the same tag: $\text{crs}' = \text{Gen}'(\text{pp}', \tau)$.
3. Output $\text{crs} = (\tau, \text{hk}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{crs}')$.

- **Prove**($\text{crs}, C, (w_1, \dots, w_k)$): On input the common reference string $\text{crs} = (\tau, \text{hk}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{crs}')$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and witnesses $w_1, \dots, w_k \in \{0, 1\}^m$, the prover algorithm proceeds as follows:

1. For each $i \in [k]$, generate the PCP proof: $\pi_i \leftarrow \text{PCP.Prove}(1^\lambda, C, i, w_i)$.
2. Commit to the proofs column-wise: for each bit-position $j \in [\ell]$, define the vector $x_j = (\pi_1[j], \dots, \pi_k[j]) \in \{0, 1\}^k$, and compute:

$$(\text{dig}_j, \sigma_{1,j}, \dots, \sigma_{k,j}) \leftarrow \text{DSEH.Hash}(\text{hk}_{\text{DSEH}}, x_j)$$

3. Let $\text{dig} = (\text{dig}_1, \dots, \text{dig}_\ell)$. Apply the correlation-intractable hash to derive the PCP randomness: $r \leftarrow \text{CI.Hash}(\text{hk}_{\text{CI}}, (C, \tau, \text{dig}))$.
4. Derive the PCP queries and state: $(Q, \text{st}) = \text{PCP.Query}(1^\lambda, C; r)$, and write $Q = (q_1, \dots, q_{|Q|})$ in a canonical order. Project the digests onto the queried positions: $\text{dig}|_Q := (\text{dig}_{q_1}, \dots, \text{dig}_{q_{|Q|}})$.
5. For each $i \in [k]$, prepare the projected PCP proof $\hat{\pi}_i = \pi_i|_Q$ and the corresponding openings $\alpha_i = (\sigma_{i,q_1}, \dots, \sigma_{i,q_{|Q|}})$, both ordered as Q is.
6. Construct the witnesses for the previous level: for each $i' \in [k/2]$, set $w'_{i'} = (\hat{\pi}_{2i'-1}, \alpha_{2i'-1}, \hat{\pi}_{2i'}, \alpha_{2i'})$, so that the branch $b \in \{0, 1\}$ of [Figure 11](#) carries $(\hat{\pi}_{i_b}, \alpha_{i_b})$.
7. Construct the intermediate circuit $\text{NewRel} = \text{NewRel}[\text{hk}_{\text{DSEH}}, \text{dig}|_Q, Q, \text{st}]$ from [Figure 11](#).
8. Compute the proof of the previous level: $\Pi' \leftarrow \text{Prove}'(\text{crs}', \text{NewRel}, (w'_1, \dots, w'_{k/2}))$.
9. Output the proof $\Pi = (\text{dig}, \Pi')$.

Constants: A dynamic somewhere-extractable hash key hk_{DSEH} , the queried digests $\text{dig}|_Q = (\text{dig}_{q_1}, \dots, \text{dig}_{q_{|Q|}})$, the queries $Q = (q_1, \dots, q_{|Q|})$, and a PCP state st .

Inputs: An index $i' \in [k/2]$ and a grouped witness $w' = (\hat{\pi}^{(0)}, \alpha^{(0)}, \hat{\pi}^{(1)}, \alpha^{(1)})$.

1. Define the original indices $i_0 = 2i' - 1$ and $i_1 = 2i'$, so that the branch $b \in \{0, 1\}$ carries the witness of the original index i_b .
2. For $b \in \{0, 1\}$:
 - For each $j \in [|Q|]$, verify the local opening: check $\text{DSEH.Verify}(\text{hk}_{\text{DSEH}}, \text{dig}_{q_j}, i_b, \hat{\pi}^{(b)}[j], \alpha^{(b)}[j]) = 1$.
 - Verify the PCP online: check $\text{PCP.Verify}(i_b, \text{st}, \hat{\pi}^{(b)}) = 1$.
3. If all checks pass, output 1. Otherwise, output 0.

Figure 11: Circuit $\text{NewRel}[\text{hk}_{\text{DSEH}}, \text{dig}|_Q, Q, \text{st}]$.

- **Verify**(crs, C, Π): On input the common reference string $\text{crs} = (\tau, \text{hk}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{crs}')$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$, and a proof $\Pi = (\text{dig}, \Pi')$, the verification algorithm proceeds as follows:
 1. Output 0 if dig is not a vector of exactly ℓ digests.
 2. Apply the correlation-intractable hash: $r \leftarrow \text{CI.Hash}(\text{hk}_{\text{CI}}, (C, \tau, \text{dig}))$.
 3. Derive the PCP queries: $(Q, \text{st}) = \text{PCP.Query}(1^\lambda, C; r)$, write $Q = (q_1, \dots, q_{|Q|})$ in the same canonical order as the prover, and project the digests onto the queried positions, $\text{dig}|_Q := (\text{dig}_{q_1}, \dots, \text{dig}_{q_{|Q|}})$.
 4. Construct the intermediate circuit $\text{NewRel} = \text{NewRel}[\text{hk}_{\text{DSEH}}, \text{dig}|_Q, Q, \text{st}]$.
 5. Delegate the bulk of the verification to the previous level by outputting $\text{Verify}'(\text{crs}', \text{NewRel}, \Pi')$.
- **Extract**(td, Π): On input the trapdoor $\text{td} = \text{td}_{\text{DSEH}}$ and a proof $\Pi = (\text{dig}, \Pi')$, the extraction algorithm proceeds as follows:
 1. Recover, bit-by-bit, the PCP proof at the index on which the hash of this level is bound: for each $j \in [\ell]$, compute:

$$b_j = \text{DSEH.Extract}(\text{td}_{\text{DSEH}}, \text{dig}_j)$$
 2. Reconstruct the target PCP proof $\pi^* = (b_1, \dots, b_\ell)$.
 3. Apply the deterministic PCP extractor: output $w \leftarrow E(\pi^*)$.

Lemma B.7 (Completeness). *Assume that Π_{PCP} is complete, that Π_{DSEH} is correct (Definition 4.1), and that Π'_{FB} is complete (Definition 4.3). Then the functional index BARG scheme Π_{FB} from Construction B.6 is complete (Definition 4.3).*

Proof. Fix $\lambda, s, s_f \in \mathbb{N}$, a tag $\tau \in \mathcal{T}$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and witnesses $w_1, \dots, w_k \in \{0, 1\}^m$ with $C(i, w_i) = 1$ for all $i \in [k]$. Let $\text{pp} = (\text{pp}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{pp}')$ be in the support of $\text{Setup}(1^\lambda, k, 1^s, 1^{s_f})$, let $\text{crs} = (\tau, \text{hk}_{\text{DSEH}}, \text{hk}_{\text{CI}}, \text{crs}') = \text{Gen}(\text{pp}, \tau)$, and let $\Pi = (\text{dig}, \Pi')$ be in the support of $\text{Prove}(\text{crs}, C, (w_1, \dots, w_k))$.

We first observe that the verifier reconstructs exactly the prover's intermediate circuit. The prover and the verifier read the same hash key hk_{DSEH} and the same tag τ out of crs , the digest vector dig is part of Π , and CI.Hash is a deterministic function of hk_{CI} and its input. Hence the verifier derives the same randomness r , the same pair $(Q, \text{st}) = \text{PCP.Query}(1^\lambda, C; r)$ and, using the same canonical ordering of Q , the same projection $\text{dig}|_Q$. The two circuits NewRel therefore agree.

We now claim that $\text{NewRel}(i', w'_{i'}) = 1$ for every $i' \in [k/2]$. Fix such an i' and a branch $b \in \{0, 1\}$, and write $i_b \in \{2i' - 1, 2i'\}$ for the corresponding original index.

- *Opening checks.* For each $j \in [|Q|]$, the prover obtained dig_{q_j} together with the openings $\sigma_{1,q_j}, \dots, \sigma_{k,q_j}$ by running $\text{DSEH.Hash}(\text{hk}_{\text{DSEH}}, x_{q_j})$ on the column $x_{q_j} = (\pi_1[q_j], \dots, \pi_k[q_j])$, and by construction $\hat{\pi}_{i_b}[j] = \pi_{i_b}[q_j]$ is the i_b^{th} block of that column, with $\alpha_{i_b}[j] = \sigma_{i_b,q_j}$ its opening. Correctness of Π_{DSEH} therefore gives $\text{DSEH.Verify}(\text{hk}_{\text{DSEH}}, \text{dig}_{q_j}, i_b, \hat{\pi}_{i_b}[j], \alpha_{i_b}[j]) = 1$.
- *Online PCP check.* The proof $\pi_{i_b} \leftarrow \text{PCP.Prove}(1^\lambda, C, i_b, w_{i_b})$ was generated for an instance with $C(i_b, w_{i_b}) = 1$, and the pair $(Q, \text{st}) = \text{PCP.Query}(1^\lambda, C; r)$ lies in the support of $\text{PCP.Query}(1^\lambda, C)$. Since the completeness of Π_{PCP} holds with probability exactly 1, it holds for *every* proof and every query pair in the support, in particular for the ones derived from r ; note that this is what allows the randomness of PCP.Query to be supplied by the correlation intractable hash rather than sampled uniformly. Hence $\text{PCP.Verify}(i_b, \text{st}, \hat{\pi}_{i_b}) = \text{PCP.Verify}(i_b, \text{st}, \pi_{i_b} | Q) = 1$.

Both checks pass for $b = 0$ and $b = 1$, so $\text{NewRel}(i', w_{i'}) = 1$ as claimed.

Finally, NewRel has size at most s_{new} by the definition of s_{new} , and $\text{crs}' = \text{Gen}'(\text{pp}', \tau)$ for pp' in the support of $\text{Setup}'(1^\lambda, k/2, 1^{s_{\text{new}}}, 1^{s'_f})$. Since $(w'_1, \dots, w'_{k/2})$ is a valid witness vector for NewRel , completeness of Π'_{FB} gives $\text{Verify}'(\text{crs}', \text{NewRel}, \Pi') = 1$, which is precisely the output of $\text{Verify}(\text{crs}, C, \Pi)$. $\square$

Lemma B.8 (Succinctness). *Suppose that Π_{DSEH} is succinct (Definition 4.1) and that Π_{PCP} has small query complexity and succinct verification (Definition B.1). Then the scheme obtained by instantiating Construction B.6 $\log k$ times on top of the base scheme of Remark B.9 is succinct (Definition 4.3).*

Proof. Fix $\lambda, k, s, s_f \in \mathbb{N}$, a tag $\tau \in \mathcal{T}$, a circuit C of size at most s and witnesses. The scheme consists of $L := \log k$ instantiations of Construction B.6, indexed so that level $j \in [0, L]$ runs on batch size $k_j := 2^j$ with parameters $(s_j, s_f^{(j)})$: level L is the outermost one and carries $(s_L, s_f^{(L)}) = (s, s_f)$, level $j - 1$ is the previous level of level j and receives the pair (s_{new}, s'_f) that level j fixes, and level 0 is the base $k_0 = 1$. Throughout we use that an efficient algorithm's output is bounded by a polynomial in the length of its input. Let $p_{\text{hk}}, p_{\text{dig}}, p_\sigma$ be the size polynomials of Definition 4.1, and let p_Q bound the query complexity of Π_{PCP} and p_{st} its state size and online verification time (Definition B.1).

The parameters grow slowly. By the choice of s'_f we have $s_f^{(j-1)} = s_f^{(j)} + O(\log k)$, so over the $L = \log k$ levels

$$s_f^{(j)} \leq F := s_f + O(\log^2 k).$$

For the circuit size, the circuit NewRel of Figure 11 hardwires a hash key, $|Q|$ digests, $|Q|$ queries and a PCP state, and performs $2|Q|$ invocations of DSEH.Verify and two of PCP.Verify ; it never evaluates C . Here $|Q| \leq p_Q(\lambda, \log s_j)$ by small query complexity; the state st and the running time of PCP.Verify are $p_{\text{st}}(\lambda, \log k, \log s_j)$ by succinct verification, the instance being an index of length at most $\log k$; and the hash key, the digests and the openings of Π_{DSEH} have length at most $p_{\text{hk}}(\lambda, \log k, s_f^{(j)}, \log s_f^{(j)}) + p_{\text{dig}}(\lambda, \log k, s_f^{(j)}, \log s_f^{(j)}) + p_\sigma(\lambda, \log k, s_f^{(j)}, \log s_f^{(j)})$, with DSEH.Verify running in time polynomial in them. Hence there is a fixed polynomial q with

$$s_{j-1} \leq q(\lambda, \log k, \log s_j, s_f^{(j)}) \leq q(\lambda, \log k, \log s_j, F).$$

Let d be any integer satisfying $d \geq q(\lambda, \log k, \log(s + d), F)$. The right-hand side grows only polylogarithmically in d , so exactly as in the proof of Lemma A.6 such a d bounded by a polynomial in $\lambda, \log k, \log s$ and s_f exists, and downward induction on j gives $s_j \leq s + d$ for every j . In particular both parameters of every level are bounded by a polynomial in $\lambda, \log k, s$ and s_f .

Each level costs ρ . The algorithm DSEH.Setup receives $(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s'_f})$, an input of length $O(\lambda + \log k + s_f^{(j)})$ since k_j is given in binary, and since it runs in polynomial time, $|\text{pp}_{\text{DSEH}}|$ is polynomial in $\lambda, \log k$ and $s_f^{(j)}$; the same holds for DSEH.SetupBinding , whose additional input is a description of size at most $s_f^{(j)}$. The algorithm Cl.Gen receives $(1^\lambda, 1^{s_{\text{Cl}}}, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}})$ in unary and runs in polynomial time, so $|\text{hk}_{\text{Cl}}|$ is polynomial in $\lambda, s_{\text{Cl}}, \ell_{\text{in}}$ and ℓ_{out} , and the same holds for Cl.GenAvoid , whose additional input is a description of size at most s_{Cl} . Now $s_{\text{Cl}} = p_{\text{BAD}}(\lambda, \log k, s_j, s_f^{(j)}, s_f^{(j)})$ by its choice above; the input length is $\ell_{\text{in}} = s_j + \ell_\tau + \ell \cdot \ell_{\text{dig}}$, where $\ell_\tau \leq \text{poly}(\lambda)$ by assumption, $\ell = \ell(\lambda, s_j)$ is the proof size

of Π_{PCP} , and $\ell_{\text{dig}} \leq p_{\text{dig}}(\lambda, \log k, s_f^{(j)}, \log s_f^{(j)})$ by the succinct digest property of Π_{DSEH} ; and the output length ℓ_{out} , the number of random bits consumed by $\text{PCP.Query}(1^\lambda, C)$ on circuits of size at most s_j , is bounded by its running time, which is polynomial in λ and s_j . Besides their calls to the previous level, the algorithms of level j parse pp , crs and Π ; run DSEH.Gen on an input of length $|\text{pp}_{\text{DSEH}}| + \ell_\tau$; evaluate $\text{Cl.Hash}(\text{hk}_{\text{Cl}}, \cdot)$ on an input of ℓ_{in} bits, in time polynomial in λ , $|\text{hk}_{\text{Cl}}|$ and ℓ_{in} ; run $\text{PCP.Query}(1^\lambda, C; r)$ in time polynomial in λ and s_j by the succinct verification property; form $\text{dig}|_Q$; and write down the circuit NewRel , of size s_{j-1} . By the previous paragraph every one of these quantities is at most $\rho(\lambda, \log k, s, s_f)$ for a single polynomial ρ serving all levels; in particular level j contributes pp_{DSEH} and hk_{Cl} to the public parameters and hk_{DSEH} , hk_{Cl} and the tag to the common reference string, of total length at most ρ in either case, and dig to the proof, of length $\ell \cdot \ell_{\text{dig}} \leq \rho$, and it runs for time at most ρ before calling the previous level.

Summing over the levels. The levels $j \in [L]$ each contribute at most ρ to $|\text{crs}|$, to $|\Pi|$ and to the running time of Verify , while level 0 has $k_0 = 1$: there $\text{crs} = \perp$, the proof is a witness $\Pi = w_1$ for a circuit of size at most s_0 , and the verifier merely evaluates that circuit. Since circuit size counts input gates (Section 3), such a circuit has at most s_0 input wires, so the witness it carries has length at most $s_0 \leq s + d$ and the evaluation takes time polynomial in $s + d$ and $|\Pi|$. Altogether $|\text{crs}| \leq \log k \cdot \rho$, which is a polynomial in λ , $\log k$, s and s_f , and we take p_{crs} to be it. A proof carries in addition the witness of the base level, so $|\Pi| \leq \log k \cdot \rho + s + d$, and we take p_Π to be it. The verifier reads its input at every level, so its running time is at most $\log k \cdot (\rho + O(|\Pi|))$ plus the evaluation of the base circuit, which is a polynomial in λ , $\log k$, s , s_f and $|\Pi|$, and we take p_{VT} to be it. These are the three bounds required by Definition 4.3. $\square$

Remark B.9 (The Base of the Recursion). At $k = 1$ the only function $f: \mathcal{T} \rightarrow [1]$ is the constant $f(\tau) = 1$, and the tag is ignored throughout. Mode indistinguishability holds perfectly, since Setup and SetupBinding induce the same (empty) distribution on pp , and $\text{Gen}(\perp, \tau) = \perp$ for every tag. Statistical somewhere extraction holds perfectly, and indeed in its strong form: if the verifier accepts then $C(1, \Pi) = 1$, and the extracted witness is $w = \Pi$, so $C(f(\tau), w) = 1$ and the experiment of Definition 4.3 never outputs 1—even when the (empty) trapdoor is handed to the adversary.

B.3 Security Proofs

Notation. Both proofs below unroll the $L := \log k$ instantiations of Construction B.6, indexed as in the proof of Lemma B.8: level $j \in [0, L]$ runs on batch size $k_j = 2^j$ with parameters $(s_j, s_f^{(j)})$ and correlation intractability parameters $(s_j^{\text{Cl}}, \ell_{\text{in}}^{(j)}, \ell_{\text{out}}^{(j)})$, level L is the outermost call and level 0 the base. We write f_j for the function of level j , so that $f_L = f$ and $f_{j-1}(\tau) = \lceil f_j(\tau)/2 \rceil$, and $\text{Setup}^{(j)}$, $\text{SetupBinding}^{(j)}$, $\text{Gen}^{(j)}$, $\text{Prove}^{(j)}$, $\text{Verify}^{(j)}$, $\text{Extract}^{(j)}$ for its algorithms, with trapdoor $\text{td}^{(j)} = \text{td}_j^{\text{DSEH}}$ and $\text{td}^{(0)} = \perp$. Thus $\text{pp}^{(0)} = \perp$ and $\text{pp}^{(j)} = (\text{pp}_j^{\text{DSEH}}, \text{hk}_j^{\text{Cl}}, \text{pp}^{(j-1)})$ for $j \in [L]$: each level $j \in [L]$ contributes a hash and a correlation intractable key, sampled either in *normal mode*,

$$\text{pp}_j^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}) \quad \text{and} \quad \text{hk}_j^{\text{Cl}} \leftarrow \text{Cl.Gen}(1^\lambda, 1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}}),$$

or in *binding mode*,

$$\begin{aligned} (\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}}) &\leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}, f_j) \\ \text{hk}_j^{\text{Cl}} &\leftarrow \text{Cl.GenAvoid}(1^\lambda, 1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}}, \text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}}). \end{aligned}$$

By construction Setup puts every level in normal mode and SetupBinding puts every level in binding mode. Given a tag τ , the common reference string of level j is $\text{crs}^{(j)} = \text{Gen}^{(j)}(\text{pp}^{(j)}, \tau) = (\tau, \text{hk}_j^{\text{DSEH}}, \text{hk}_j^{\text{Cl}}, \text{crs}^{(j-1)})$, where $\text{hk}_j^{\text{DSEH}} = \text{DSEH.Gen}(\text{pp}_j^{\text{DSEH}}, \tau)$ and $\text{crs}^{(0)} = \perp$. The relation $\text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}}$ hardwires the hash trapdoor of its own level, so it is well defined exactly when the hash of level j is binding; this is why, within a level, the hash is switched before the correlation intractable key. No relation depends on the trapdoor of another level, so across levels the two modes may be mixed in any pattern.

Proofs structure. Both proofs walk through the L levels in a single sequence of hybrids, rather than invoking a one-level statement recursively. The reason is that a recursive invocation would apply the guarantee of an ingredient afresh at every level and so produce one negligible function per level; since the number of levels grows with λ , the maximum

of $L(\lambda)$ negligible functions need not be negligible. Instead, in each proof we bound the contribution of *all* the levels at once by exhibiting a single adversary that samples the level it attacks, exactly as in [Theorem 5.7](#) and [Theorem 5.10](#).

Lemma B.10 (Mode Indistinguishability). *Suppose that Π_{DSEH} satisfies mode indistinguishability ([Definition 4.1](#)) and that Π_{CI} satisfies mode indistinguishability ([Definition B.3](#)). Then [Construction B.6](#) satisfies mode indistinguishability ([Definition 4.3](#)).*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game of [Definition 4.3](#) against [Construction B.6](#). We first fix its first message. The advantage of $\mathcal{A}$ is the average, over its first message, of its advantage conditioned on that message, so for every λ there is a first message $k, 1^s, 1^{sf}$ together with a description of f that maximizes it. It suffices to bound the advantage of the adversary that sends this message and then runs the second stage of $\mathcal{A}$, so we assume from here on that the first message is fixed and write $L = \log k$. Since $\mathcal{A}$ is efficient and writes k in binary, $L \leq \text{poly}(\lambda)$. The point of fixing the first message is that L is now a fixed function of λ and not a random variable, so a reduction may sample a level $j \xleftarrow{R} [L]$ before it runs $\mathcal{A}$. If $L = 0$ then $k = 1$ and [Construction B.6](#) is the trivial scheme of [Remark B.9](#), whose two modes induce the same distribution on pp , so the advantage of $\mathcal{A}$ is 0; we therefore assume $L \geq 1$ from here on.

All the experiments below differ only in how the challenger samples the public parameters $\text{pp}^{(L)}$ that it sends to $\mathcal{A}$; in each of them the output of the experiment is the guess b' that $\mathcal{A}$ outputs, and no trapdoor is ever given to $\mathcal{A}$. The goal is to prove that $\mathcal{A}$ cannot distinguish between the two following experiments except with negligible probability:

- $\text{Real}^{(0)}$: the mode indistinguishability game with $b = 0$, in which every level is in normal mode. Formally:

1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^s, 1^{sf}$ and the description of f .
2. The challenger runs $\text{Setup}^{(L)}(1^\lambda, k, 1^s, 1^{sf})$. That is, for each level $j = 1, \dots, L$:
 - Sample the hash parameters $\text{pp}_j^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, k_j, 1^{sf_j}, 1^{sf_j})$.
 - Sample the correlation intractable hash key $\text{hk}_j^{\text{CI}} \leftarrow \text{CI.Gen}(1^\lambda, 1^{sf_j}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}})$.

The challenger assembles $\text{pp}^{(L)}$ from these components and $\text{pp}^{(0)} = \perp$, and sends it to $\mathcal{A}$.

3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.

- $\text{Real}^{(1)}$: the mode indistinguishability game with $b = 1$, in which every level is in binding mode. This experiment is the same as $\text{Real}^{(0)}$, except that the challenger runs $\text{SetupBinding}^{(L)}(1^\lambda, k, 1^s, 1^{sf}, f)$, that is, for each level $j = 1, \dots, L$ it samples $(\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_j, 1^{sf_j}, 1^{sf_j}, f_j)$ and then $\text{hk}_j^{\text{CI}} \leftarrow \text{CI.GenAvoid}(1^\lambda, 1^{sf_j}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}}, \text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}})$, in this order.

To do so, we define the following sequence of hybrids for each $j \in [0, L]$.

- Hyb_j : in this experiment the levels $\ell \leq j$ are in binding mode, whereas the levels $\ell > j$ are in normal mode. Formally:

1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^s, 1^{sf}$ and the description of f .
2. The challenger samples the following. For each level $\ell = 1, \dots, L$, if $\ell \leq j$:
 - Sample $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_\ell, 1^{sf_\ell}, 1^{sf_\ell}, f_\ell)$.
 - Sample $\text{hk}_\ell^{\text{CI}} \leftarrow \text{CI.GenAvoid}(1^\lambda, 1^{sf_\ell}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}}, \text{BAD}_{\lambda, f_\ell, \text{td}_\ell^{\text{DSEH}}})$.

If $\ell > j$:

- Sample $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, k_\ell, 1^{sf_\ell}, 1^{sf_\ell})$.
- Sample $\text{hk}_\ell^{\text{CI}} \leftarrow \text{CI.Gen}(1^\lambda, 1^{sf_\ell}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}})$.

The challenger assembles $\text{pp}^{(L)}$ and sends it to $\mathcal{A}$.

3. The adversary $\mathcal{A}$ outputs b' , the output of the experiment.

In particular, Hyb_0 is identical to $\text{Real}^{(0)}$, since level 0 contributes no components, and Hyb_L is identical to $\text{Real}^{(1)}$. The relation $\text{BAD}_{\lambda, f_\ell, \text{td}_\ell^{\text{DSEH}}}$ hardwires the hash trapdoor of level ℓ , so it is well defined in Hyb_j for every level $\ell \leq j$, which is in binding mode. We also define an intermediate hybrid for each $j \in [L]$:

- Hyb'_j : the same as Hyb_{j-1} , except that for level j the hash is sampled in binding mode while the correlation intractable key is still in normal mode. Formally:

2. For level j , the challenger samples $(\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}, f_j)$ and $\text{hk}_j^{\text{Cl}} \leftarrow \text{Cl.Gen}(1^\lambda, 1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}})$. The rest of the experiment remains unchanged.

The chain of experiments is

$$\text{Hyb}_0, \text{Hyb}'_1, \text{Hyb}_1, \text{Hyb}'_2, \dots, \text{Hyb}_{L-1}, \text{Hyb}'_L, \text{Hyb}_L,$$

where a step from Hyb_{j-1} to Hyb'_j switches the hash of level j and a step from Hyb'_j to Hyb_j switches the correlation intractable key of that level. We bound the two kinds of steps separately. In both claims we bound the *signed* sum over the levels rather than each level separately: this is what a single reduction that samples the level can bound, whereas bounding each level separately would again require one negligible function per level, and the number of levels grows with λ .

Claim B.11. *If Π_{DSEH} satisfies mode indistinguishability, then there exists a negligible function $\text{negl}^{\text{DSEH}}$ such that*

$$\left| \sum_{j \in [L]} \left(\Pr[\text{Hyb}_{j-1}(\mathcal{A}) = 1] - \Pr[\text{Hyb}'_j(\mathcal{A}) = 1] \right) \right| \leq L \cdot \text{negl}^{\text{DSEH}}(\lambda).$$

Proof. Consider the following efficient adversary $\mathcal{B}$ for the mode indistinguishability game of Π_{DSEH} (Definition 4.1), which takes a level $j \in [L]$ as an input.

1. Run $\mathcal{A}(1^\lambda)$ to obtain $k, 1^s, 1^{s_f}$ and the description of f , set $L = \log k$, and compute the level parameters $(k_\ell, s_\ell, s_f^{(\ell)})$, the correlation intractability parameters $(s_\ell^{\text{Cl}}, \ell_{\text{in}}^{(\ell)}, \ell_{\text{out}}^{(\ell)})$ and the functions f_ℓ for every $\ell \in [L]$.
2. For each $\ell = 1, \dots, j-1$, sample $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_\ell, 1^{s_f^{(\ell)}}, 1^{s_f^{(\ell)}}, f_\ell)$ and then $\text{hk}_\ell^{\text{Cl}} \leftarrow \text{Cl.GenAvoid}(1^\lambda, 1^{s_\ell^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}}, \text{BAD}_{\lambda, f_\ell, \text{td}_\ell^{\text{DSEH}}})$, in this order, so that the relation of level ℓ has the trapdoor of that level's hash available.
3. Send $k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}$ and the description of f_j to the challenger, and let $\text{pp}_j^{\text{DSEH}}$ be the public parameters it returns. Sample $\text{hk}_j^{\text{Cl}} \leftarrow \text{Cl.Gen}(1^\lambda, 1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}})$.
4. For each $\ell = j+1, \dots, L$, sample $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, k_\ell, 1^{s_f^{(\ell)}}, 1^{s_f^{(\ell)}})$ and $\text{hk}_\ell^{\text{Cl}} \leftarrow \text{Cl.Gen}(1^\lambda, 1^{s_\ell^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}})$.
5. Assemble $\text{pp}^{(L)}$ from these components and $\text{pp}^{(0)} = \perp$, send it to $\mathcal{A}$, and output whatever $\mathcal{A}$ outputs.

The adversary $\mathcal{B}$ is efficient: it runs $\mathcal{A}$ once and samples at most $2L \leq \text{poly}(\lambda)$ further keys, each on inputs bounded by $\text{poly}(\lambda, \log k, s, s_f)$ by the parameter growth established in the proof of Lemma B.8. It is admissible because f_j has description size and depth at most $s_f^{(j)}$, which are the bounds it sends to the challenger.

The simulation is perfect in both modes. Every level $\ell < j$ is sampled in binding mode and every level $\ell > j$ in normal mode, in both cases as the challengers of Hyb_{j-1} and of Hyb'_j do, and the correlation intractable key of level j is sampled in normal mode, as in both experiments. The only difference is the hash of level j : if the mode indistinguishability challenger is in mode $b = 0$ then $\text{pp}_j^{\text{DSEH}}$ is distributed as $\text{DSEH.Setup}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}})$ and $\mathcal{B}(j)$ perfectly simulates Hyb_{j-1} , and if it is in mode $b = 1$ then $\text{pp}_j^{\text{DSEH}}$ is distributed as the first output of $\text{DSEH.SetupBinding}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}, f_j)$ and $\mathcal{B}(j)$ perfectly simulates Hyb'_j . Note that $\mathcal{B}$ never uses $\text{td}_j^{\text{DSEH}}$, which

the challenger does not give it, and which the two experiments do not use either: the correlation intractable key of level j is normal in both. Consequently, for every level j ,

$$\Pr[\mathcal{B}(j) = 1 : b = 0] - \Pr[\mathcal{B}(j) = 1 : b = 1] = \Pr[\text{Hyb}_{j-1}(\mathcal{A}) = 1] - \Pr[\text{Hyb}'_j(\mathcal{A}) = 1].$$

Let $\mathcal{B}^*$ be the adversary that samples $j \xleftarrow{\mathcal{R}} [L]$ and runs $\mathcal{B}(j)$; this is well defined because L was fixed at the start of the proof. It is a single efficient adversary, so by mode indistinguishability of Π_{DSEH} there is a negligible function $\text{negl}^{\text{DSEH}}$ with

$$|\Pr[\mathcal{B}^* = 1 : b = 0] - \Pr[\mathcal{B}^* = 1 : b = 1]| \leq \text{negl}^{\text{DSEH}}(\lambda).$$

Since j is sampled uniformly and independently of the challenger's bit, the left-hand side equals

$$\left| \frac{1}{L} \sum_{j \in [L]} \left(\Pr[\mathcal{B}(j) = 1 : b = 0] - \Pr[\mathcal{B}(j) = 1 : b = 1] \right) \right| = \frac{1}{L} \cdot \left| \sum_{j \in [L]} \left(\Pr[\text{Hyb}_{j-1}(\mathcal{A}) = 1] - \Pr[\text{Hyb}'_j(\mathcal{A}) = 1] \right) \right|,$$

and the claim follows by multiplying by L . $\square$

Claim B.12. *If Π_{Cl} satisfies mode indistinguishability, then there exists a negligible function negl^{Cl} such that*

$$\left| \sum_{j \in [L]} \left(\Pr[\text{Hyb}'_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_j(\mathcal{A}) = 1] \right) \right| \leq L \cdot \text{negl}^{\text{Cl}}(\lambda).$$

Proof. Consider the following efficient adversary C for the mode indistinguishability game of Π_{Cl} (Definition B.3), which takes a level $j \in [L]$ as an input.

1. Run $\mathcal{A}(1^\lambda)$ to obtain $k, 1^s, 1^{s_f}$ and the description of f , set $L = \log k$, and compute the level parameters and the functions f_ℓ as before.
2. For each $\ell = 1, \dots, j-1$, sample level ℓ in binding mode, that is, $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_\ell, 1^{s_f^{(\ell)}}, 1^{s_f^{(\ell)}}, f_\ell)$ and then $\text{hk}_\ell^{\text{Cl}} \leftarrow \text{Cl.GenAvoid}(1^\lambda, 1^{s_\ell^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}}, \text{BAD}_{\lambda, f_\ell, \text{td}_\ell^{\text{DSEH}}})$.
3. Sample $(\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}, f_j)$, which is the mode of the hash of level j in both experiments, and construct the description of the relation $\text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}}$ of Definition B.4.
4. Send $1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}}$ and the description of $\text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}}$ to the challenger, and let hk_j^{Cl} be the hash key it returns.
5. For each $\ell = j+1, \dots, L$, sample level ℓ in normal mode, that is, $\text{pp}_\ell^{\text{DSEH}} \leftarrow \text{DSEH.Setup}(1^\lambda, k_\ell, 1^{s_f^{(\ell)}}, 1^{s_f^{(\ell)}})$ and $\text{hk}_\ell^{\text{Cl}} \leftarrow \text{Cl.Gen}(1^\lambda, 1^{s_\ell^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}})$.
6. Assemble $\text{pp}^{(L)}$, send it to $\mathcal{A}$, and output whatever $\mathcal{A}$ outputs.

As before, C is efficient and admissible, and unlike $\mathcal{B}$ it does hold the trapdoor $\text{td}_j^{\text{DSEH}}$ that naming the relation requires, because it sampled the hash of level j itself. The simulation is perfect in both modes: all the levels other than j are sampled exactly as in the two experiments, the hash of level j is binding in both of them, and the only difference is the correlation intractable key of level j . If the challenger is in mode $b = 0$ then hk_j^{Cl} is distributed as $\text{Cl.Gen}(1^\lambda, 1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}})$ and $C(j)$ perfectly simulates Hyb'_j , and if it is in mode $b = 1$ then hk_j^{Cl} is distributed as $\text{Cl.GenAvoid}(1^\lambda, 1^{s_j^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}}, \text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}})$, so that level j is in binding mode in full and $C(j)$ perfectly simulates Hyb_j ; recall that in Hyb_j every level $\ell \leq j$ is in binding mode, which is the case here, and every level $\ell > j$ is in normal mode, which is the case by the fifth step. Hence, for every level j ,

$$\Pr[C(j) = 1 : b = 0] - \Pr[C(j) = 1 : b = 1] = \Pr[\text{Hyb}'_j(\mathcal{A}) = 1] - \Pr[\text{Hyb}_j(\mathcal{A}) = 1],$$

and the claim follows exactly as in Claim B.11, by applying mode indistinguishability of Π_{Cl} to the adversary C^* that samples $j \xleftarrow{\mathcal{R}} [L]$ and runs $C(j)$. $\square$

Combining [Claim B.11](#) and [Claim B.12](#) with the triangle inequality along the chain,

$$|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_L(\mathcal{A}) = 1]| \leq L \cdot (\text{negl}^{\text{DSEH}}(\lambda) + \text{negl}^{\text{Cl}}(\lambda)),$$

which is negligible because $L \leq \text{poly}(\lambda)$. Since Hyb_0 is $\text{Real}^{(0)}$ and Hyb_L is $\text{Real}^{(1)}$, this is exactly mode indistinguishability of [Construction B.6](#). $\square$

Lemma B.13 (Somewhere Extraction). *Suppose that Π_{DSEH} satisfies somewhere extraction ([Definition 4.1](#)) and that Π_{Cl} is somewhere statistically correlation intractable for the family BAD of [Definition B.4](#), defined for a PCP scheme Π_{PCP} with fast online verification that is a proof of knowledge. Then [Construction B.6](#) satisfies somewhere extraction ([Definition 4.3](#)).*

Proof. Let $\mathcal{A}$ be an unbounded adversary for the somewhere extraction game of [Definition 4.3](#) against [Construction B.6](#). Since the adversary is unbounded, we assume without loss of generality that it is deterministic; in particular its first message, and hence the number of levels $L = \log k$, is a fixed function of λ , which is what makes the level sampling below well defined. Since [Definition 4.3](#) bounds $k \leq 2^\lambda$, we have $L \leq \lambda$. If $L = 0$ then $k = 1$ and [Construction B.6](#) is the trivial scheme of [Remark B.9](#), whose somewhere extraction experiment never outputs 1 ([Claim B.16](#) below); we therefore assume $L \geq 1$ from here on.

We first write the experiment out in full, with the algorithms of [Construction B.6](#) substituted in, and introduce along the way the notation we use below.

1. On input 1^λ , the adversary $\mathcal{A}$ outputs $k, 1^s, 1^{s_f}$ and the description of a function $f: \mathcal{T} \rightarrow [k]$ of description size at most s_f .
2. The challenger runs $\text{SetupBinding}^{(L)}(1^\lambda, k, 1^s, 1^{s_f}, f)$, that is, it puts every level in binding mode. For each $\ell = 1, \dots, L$ it samples

$$\begin{aligned} (\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) &\leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_\ell, 1^{s_f^{(\ell)}}, 1^{s_f^{(\ell)}}, f_\ell) \\ \text{hk}_\ell^{\text{Cl}} &\leftarrow \text{Cl.GenAvoid}(1^\lambda, 1^{s_\ell^{\text{Cl}}}, 1^{\ell_{\text{in}}^{(\ell)}}, 1^{\ell_{\text{out}}^{(\ell)}}, \text{BAD}_{\lambda, f_\ell, \text{td}_\ell^{\text{DSEH}}}). \end{aligned}$$

It assembles $\text{pp}^{(L)}$ and sends it to $\mathcal{A}$, keeping $\text{td}^{(L)} = \text{td}_L^{\text{DSEH}}$.

3. The adversary $\mathcal{A}$ outputs a tag $\tau \in \mathcal{T}$, a circuit $C: [k] \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and a proof Π .
4. The challenger derives the common reference string $\text{crs}^{(L)} = \text{Gen}^{(L)}(\text{pp}^{(L)}, \tau)$, which fixes the hash key $\text{hk}_\ell^{\text{DSEH}} = \text{DSEH.Gen}(\text{pp}_\ell^{\text{DSEH}}, \tau)$ and the string $\text{crs}^{(\ell)} = (\tau, \text{hk}_\ell^{\text{DSEH}}, \text{hk}_\ell^{\text{Cl}}, \text{crs}^{(\ell-1)})$ of every level $\ell \in [L]$.
5. The challenger computes the following quantities. They do not affect the output of the experiment, and are used only to define the hybrids below. Set $C^{(L)} := C$ and $\Pi^{(L)} := \Pi$, and for each level $\ell = L, L-1, \dots, 1$, in this order:
 - The index $i^{(\ell)} := f_\ell(\tau)$ on which the hash and the scheme of that level are bound.
 - The parse $\Pi^{(\ell)} = (\text{dig}^{(\ell)}, \Pi^{(\ell-1)})$ of the proof of level ℓ , where $\text{dig}^{(\ell)}$ is a vector of digests, one for each bit of a PCP proof for the circuit $C^{(\ell)}$, and we write $\text{dig}_n^{(\ell)}$ for its n^{th} entry.
 - The target PCP proof of level ℓ , namely $\pi^{*(\ell)}$ whose n^{th} bit is $\text{DSEH.Extract}(\text{td}_\ell^{\text{DSEH}}, \text{dig}_n^{(\ell)})$, and the extracted witness $w^{(\ell)} := \text{Extract}^{(\ell)}(\text{td}^{(\ell)}, \Pi^{(\ell)}) = E(\pi^{*(\ell)})$, where E is the PCP extractor.
 - The randomness $r^{(\ell)} := \text{Cl.Hash}(\text{hk}_\ell^{\text{Cl}}, (C^{(\ell)}, \tau, \text{dig}^{(\ell)}))$, the PCP queries and associated state $(Q^{(\ell)}, \text{st}^{(\ell)}) := \text{PCP.Query}(1^\lambda, C^{(\ell)}; r^{(\ell)})$, and the circuit of the level below,

$$C^{(\ell-1)} := \text{NewRel}[\text{hk}_\ell^{\text{DSEH}}, \text{dig}^{(\ell)}|_{Q^{(\ell)}}, Q^{(\ell)}, \text{st}^{(\ell)}],$$

which is the circuit that the verifier of level ℓ hands to the verifier of level $\ell - 1$ ([Figure 11](#)).

At level 0 we also set $w^{(0)} := \text{Extract}^{(0)}(\text{td}^{(0)}, \Pi^{(0)}) = \Pi^{(0)}$, as prescribed by [Remark B.9](#).

6. The experiment outputs 1 if and only if $\text{Verify}^{(L)}(\text{crs}^{(L)}, C, \Pi) = 1$ and $C(f(\tau), w^{(L)}) \neq 1$.

Two features of [Construction B.6](#) drive the whole argument. First, the verifier of level ℓ delegates its decision to the verifier of the level below: it accepts if and only if $\text{Verify}^{(\ell-1)}(\text{crs}^{(\ell-1)}, C^{(\ell-1)}, \Pi^{(\ell-1)}) = 1$. Second, $\text{Extract}^{(\ell)}$ reads the target PCP proof out of the digests with the hash trapdoor of level ℓ and then applies the PCP extractor, so $w^{(\ell)} = E(\pi^{*(\ell)})$.

The main idea is to walk down the levels of the recursion, showing that if extraction fails at level ℓ then it already fails at level $\ell - 1$, unless the correlation intractable hash of level ℓ failed to avoid its bad relation. We therefore define, for each $j \in [0, L]$, the hybrid experiment

- Hyb_j : identical to the experiment above in its first five steps, but its output is computed at level j instead of level L . That is, it outputs 1 if and only if

$$\text{Verify}^{(j)}(\text{crs}^{(j)}, C^{(j)}, \Pi^{(j)}) = 1 \quad \text{and} \quad C^{(j)}(f_j(\tau), w^{(j)}) \neq 1.$$

In particular Hyb_L is the somewhere extraction experiment of [Definition 4.3](#) for [Construction B.6](#). We also define, for each $j \in [L]$, the event

$$\text{Bad}_j := \left\{ ((C^{(j)}, \tau, \text{dig}^{(j)}), r^{(j)}) \in \text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}} \right\},$$

that is, the event that the correlation intractable hash of level j maps its input into the bad relation of that level.

Claim B.14. *If Π_{DSEH} satisfies somewhere extraction, then for every $j \in [L]$, we have*

$$\Pr[\text{Hyb}_j(\mathcal{A}) = 1] \leq \Pr[\text{Hyb}_{j-1}(\mathcal{A}) = 1] + \Pr[\text{Bad}_j].$$

Proof. It suffices to show that on every execution in which Hyb_j outputs 1, either Hyb_{j-1} outputs 1 or Bad_j occurs. So consider an execution in which Hyb_j outputs 1 and Hyb_{j-1} outputs 0; we show that Bad_j occurs.

Since Hyb_j outputs 1, the verifier of level j accepts and, by construction, so does the verifier of level $j - 1$ on $(C^{(j-1)}, \Pi^{(j-1)})$. As Hyb_{j-1} outputs 0, its second condition must therefore fail, that is,

$$C^{(j-1)}(f_{j-1}(\tau), w^{(j-1)}) = 1.$$

Write $i' := f_{j-1}(\tau)$ and parse $w^{(j-1)} = (\hat{\pi}^{(0)}, \alpha^{(0)}, \hat{\pi}^{(1)}, \alpha^{(1)})$ as in [Figure 11](#). Since $f_{j-1}(\tau) = \lceil f_j(\tau)/2 \rceil$, we have $i^{(j)} \in \{2i' - 1, 2i'\}$, so there is a unique branch $b^* \in \{0, 1\}$ with $i_{b^*} = i^{(j)}$, where $i_0 = 2i' - 1$ and $i_1 = 2i'$.

Because $C^{(j-1)}(i', w^{(j-1)}) = 1$, all the local opening checks of branch b^* pass: for every $n \in [|Q^{(j)}|]$, writing q_n for the n^{th} query of $Q^{(j)}$,

$$\text{DSEH.Verify}(\text{hk}_j^{\text{DSEH}}, \text{dig}_{q_n}^{(j)}, i^{(j)}, \hat{\pi}^{(b^*)}[n], \alpha^{(b^*)}[n]) = 1.$$

The pair $(\text{pp}_j^{\text{DSEH}}, \text{td}_j^{\text{DSEH}})$ is in the support of $\text{DSEH.SetupBinding}(1^\lambda, k_j, 1^{s_f^{(j)}}, 1^{s_f^{(j)}}, f_j)$ and the hash key $\text{hk}_j^{\text{DSEH}} = \text{DSEH.Gen}(\text{pp}_j^{\text{DSEH}}, \tau)$ is bound on the index $i^{(j)} = f_j(\tau)$, so the perfect somewhere extraction of Π_{DSEH} gives $\text{DSEH.Extract}(\text{td}_j^{\text{DSEH}}, \text{dig}_{q_n}^{(j)}) = \hat{\pi}^{(b^*)}[n]$ for every n . By the definition of the target PCP proof, $\pi^{*(j)}[q_n] = \text{DSEH.Extract}(\text{td}_j^{\text{DSEH}}, \text{dig}_{q_n}^{(j)})$, hence

$$\hat{\pi}^{(b^*)} = \pi^{*(j)}|_{Q^{(j)}}.$$

Since $C^{(j-1)}(i', w^{(j-1)}) = 1$, the online PCP check of branch b^* passes ([Figure 11](#)), that is, $\text{PCP.Verify}(i^{(j)}, \text{st}^{(j)}, \hat{\pi}^{(b^*)}) = 1$, and substituting the display above gives

$$\text{PCP.Verify}(i^{(j)}, \text{st}^{(j)}, \pi^{*(j)}|_{Q^{(j)}}) = 1.$$

On the other hand, Hyb_j outputs 1, so $C^{(j)}(i^{(j)}, w^{(j)}) \neq 1$ with $w^{(j)} = E(\pi^{*(j)})$. These two facts together are precisely the membership condition of $\text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}}$ ([Definition B.4](#)) for the pair $((C^{(j)}, \tau, \text{dig}^{(j)}), r^{(j)})$, since $(Q^{(j)}, \text{st}^{(j)})$ is derived from $C^{(j)}$ and $r^{(j)}$ exactly as that definition prescribes. Hence Bad_j occurs, as required. $\square$

Claim B.15. *If Π_{CI} satisfies statistical correlation intractability for the family BAD of [Definition B.4](#), then there exists a negligible function negl^{CI} such that $\sum_{j \in [L]} \Pr[\text{Bad}_j] \leq L \cdot \text{negl}^{\text{CI}}(\lambda)$.*

Proof. Consider the following unbounded adversary $\mathcal{B}$ for the somewhere statistical correlation intractability game of Π_{CI} (Definition B.3):

1. Run $\mathcal{A}(1^\lambda)$ to obtain $k, 1^s, 1^{sf}$ and the description of f , set $L = \log k$, compute the level parameters and the functions f_ℓ , and sample $j \xleftarrow{\mathbb{R}} [L]$.
2. For every $\ell \in [L]$, sample $(\text{pp}_\ell^{\text{DSEH}}, \text{td}_\ell^{\text{DSEH}}) \leftarrow \text{DSEH.SetupBinding}(1^\lambda, k_\ell, 1^{s_\ell^{(f)}}, 1^{sf^{(f)}}, f_\ell)$, and for every $\ell \neq j$ sample $\text{hk}_\ell^{\text{CI}} \leftarrow \text{CI.GenAvoid}(1^\lambda, 1^{s_\ell^{\text{CI}}}, 1^{\ell_{\text{in}}^{(f)}}, 1^{\ell_{\text{out}}^{(f)}}, \text{BAD}_{\lambda, f_\ell, \text{td}_\ell^{\text{DSEH}}})$.
3. Construct the description of $\text{BAD}_{\lambda, f_j, \text{td}_j^{\text{DSEH}}}$ and send $1^{s_j^{\text{CI}}}, 1^{\ell_{\text{in}}^{(j)}}, 1^{\ell_{\text{out}}^{(j)}}$ together with it to the challenger, which replies with an avoiding key hk_j^{CI} .
4. Assemble $\text{pp}^{(L)}$, send it to $\mathcal{A}$, and let (τ, C, Π) be its output.
5. Derive $\text{crs}^{(L)} = \text{Gen}^{(L)}(\text{pp}^{(L)}, \tau)$ and form the quantities $C^{(L)}, \dots, C^{(j)}$ and $\Pi^{(L)}, \dots, \Pi^{(j)}$ of the fourth and fifth steps of the experiment, and output $x = (C^{(j)}, \tau, \text{dig}^{(j)})$.

The adversary $\mathcal{B}$ is admissible, and it holds every trapdoor it needs: it samples all the hashes itself, so it can name the relation of every level, including the one it submits. The public parameters it hands $\mathcal{A}$ are distributed exactly as in the experiment, since every level other than j is sampled in binding mode by $\mathcal{B}$ and level j has its hash sampled by $\mathcal{B}$ and its avoiding key supplied by the challenger. Therefore $(C^{(j)}, \tau, \text{dig}^{(j)})$ and $r^{(j)} = \text{CI.Hash}(\text{hk}_j^{\text{CI}}, x)$ are distributed as in the experiment, so conditioned on the sampled level being j , the adversary $\mathcal{B}$ wins if and only if Bad_j occurs. Note that $\mathcal{B}$ runs $\mathcal{A}$ once and then performs $L - j$ rounds of local post-processing; it does not invoke a reduction recursively.

By the somewhere statistical correlation intractability of Π_{CI} there is a negligible function negl^{CI} with $\Pr[\mathcal{B} \text{ wins}] \leq \text{negl}^{\text{CI}}(\lambda)$, and since $\mathcal{B}$ samples j uniformly and independently of everything else,

$$\Pr[\mathcal{B} \text{ wins}] = \frac{1}{L} \sum_{j \in [L]} \Pr[\text{Bad}_j].$$

Multiplying by L gives the claim. $\square$

Claim B.16. $\Pr[\text{Hyb}_0(\mathcal{A}) = 1] = 0$.

Proof. The scheme of level 0 is the trivial one of Remark B.9: there $k_0 = 1$, a proof is the witness itself, the verifier of level 0 outputs $C^{(0)}(1, \Pi^{(0)})$, and $\text{Extract}^{(0)}$ outputs $\Pi^{(0)}$, so $w^{(0)} = \Pi^{(0)}$. Hence if the verifier of level 0 accepts then $C^{(0)}(1, w^{(0)}) = 1$, while $f_0(\tau) = 1$ is the only index available, so the two conditions of Hyb_0 cannot hold simultaneously. $\square$

Chaining Claim B.14 down from $j = L$ to $j = 1$ and using Claim B.16,

$$\Pr[\text{Hyb}_L(\mathcal{A}) = 1] \leq \Pr[\text{Hyb}_0(\mathcal{A}) = 1] + \sum_{j \in [L]} \Pr[\text{Bad}_j] = \sum_{j \in [L]} \Pr[\text{Bad}_j] \leq L \cdot \text{negl}^{\text{CI}}(\lambda) \leq \lambda \cdot \text{negl}^{\text{CI}}(\lambda)$$

by Claim B.15, using $L \leq \lambda$ in the last step. The right-hand side is negligible, and since Hyb_L is the somewhere extraction experiment for Construction B.6, this is exactly somewhere extraction. $\square$

B.4 Putting It Together

Theorem B.17 (Functional Index BARG). *Assume there exists a PCP with fast online verification for C-SAT that is a proof of knowledge, a dynamic somewhere-extractable hash for the block space $\{0, 1\}$ and the tag space $\mathcal{T}$ (Definition 4.1), and a hash function that is somewhere statistically correlation intractable for the family of relations BAD of Definition B.4. Then there exists a functional index BARG with statistical extraction for the tag space $\mathcal{T}$, as per Definition 4.3.*

Proof. We show that Construction B.6, instantiated with the three ingredients of the hypothesis, satisfies the four properties of Definition 4.3. Completeness holds by Lemma B.7, mode indistinguishability by Lemma B.10, somewhere extraction by Lemma B.13, and succinctness by Lemma B.8. $\square$