# Monotone-Policy Aggregate Signatures

Maya Farber Brodsky[1], Arka Rai Choudhuri[2], Abhishek Jain[3], and Omer Paneth[1]

[1]Tel Aviv University
[2]NTT Research
[3]NTT Research and JHU

**Abstract**

The notion of aggregate signatures allows for combining signatures from different parties into a short certificate that attests that *all* parties signed a message. In this work, we lift this notion to capture different, more expressive signing policies. For example, we can certify that a message was signed by a (weighted) threshold of signers.

We present the first constructions of aggregate signatures for monotone policies based on standard polynomial-time cryptographic assumptions. The aggregate signatures in our schemes are succinct, i.e., their size is *independent* of the number of signers. Moreover, verification is also succinct if all parties sign the same message (or if the messages have a succinct representation). All prior work requires either interaction between the parties or non-standard assumptions (that imply SNARKs for NP).

Our signature schemes are based on non-interactive batch arguments (BARGs) for monotone policies [Brakerski-Brodsky-Kalai-Lombardi-Paneth, Crypto'23]. In contrast to previous constructions, our BARGs satisfy a new notion of *adaptive* security which is instrumental to our application. Our new BARGs for monotone policies can be constructed from standard BARGs and other standard assumptions.

# Contents

# 1 Introduction

Suppose that a group of parties wish to jointly sign a message $m$ such that anyone can verify that a majority of the parties signed $m$. Such a scenario is commonplace in applications such as byzantine agreement [Tou84], crypto wallets [sep21] and many other settings involving decentralization of trust [DFI, Chi, EJN17, dra17, Pol20].

Consider the following notion of digital signatures that captures several properties desirable for such applications: each of the $k$ parties locally computes and publishes a verification key for its preferred signature scheme and uses the corresponding secret key to sign messages. Later, an untrusted party called the "aggregator" can combine the signatures of multiple parties into a joint signature $\sigma$ to certify that at least $t$ out of $k$ parties signed a message. The combined signature $\sigma$ can be verified given a digest of all $k$ verification keys and the vector of messages signed (but without knowing the identity of the $t$ signers). The key requirement is succinctness:

- The size of the aggregate signature should be *sublinear* in $k$ and even $t$, and ideally only depend on the security parameter.
- If all parties sign the *same* message (or, more generally, if the vector of messages has some succinct representation), the verification time should also be *sublinear* in $t$.
- Ideally, the aggregation time should grow with the number of signatures $t$ as opposed to the number of parties $k$.

The security requirement is that no polynomial-time adversary that corrupts less than $t$ parties in an *adaptive* manner should be able to create valid signatures.

This notion combines the best features of two central notions of multiparty signatures: threshold signatures [Des88, DF90] and aggregate signatures [BGLS03, IN83].[1] Threshold signatures, widely used in the blockchain ecosystem, support threshold signing with succinct verification but require *interactive* protocols for key setup and (in most cases) signing. Aggregate signatures dispense with the necessity of interaction but can only attest that *all* $k$ parties signed (i.e., they do not support threshold signing). Therefore, the above notion can be viewed as a threshold variant of aggregate signatures or as an ad-hoc variant of threshold signatures. To see the appeal of this notion, consider decentralized autonomous organizations (DAO) where members vote on proposals to make joint decisions on governance of assets. Threshold signatures are a natural cryptographic tool for implementing such voting; however, running an interactive key setup between all members is unrealistic. Our notion eliminates this barrier.

**Monotone-Policy Aggregate Signatures.** We can generalize the above notion to capture more expressive signing policies. For example, consider weighted thresholds, where each party has a different weight and the signing policy is defined with respect to the sum of the party weights. Such a policy is useful in proof-of-stake blockchains [NHN+19]. Another example is threshold of thresholds capturing users with some hierarchical structure. In general, the signing policy might be described by a monotone function that takes as input a list of signers and decides whether they are authorized. We refer to this primitive as monotone-policy aggregate signatures.

[1] In the literature, there are two distinct notions of signature schemes that support signature aggregation: multisignatures [IN83], where all parties sign the same message, and aggregate signatures [BGLS03], where parties may sign distinct messages. For simplicity of exposition, we use the terminology of aggregate signatures to refer to both settings.

It is not difficult to see that this primitive can be realized by combining standard signature schemes with (adaptively sound) succinct non-interactive arguments of knowledge (SNARKs) for NP [Mic94, BCC+17]. The aggregate signature on a message $m$ simply consists of a SNARK that proves the existence of valid signatures on $m$ from an authorized set of signers. The succinctness property of the SNARK translates to the succinctness of aggregate signature. Sublinear verification time can be achieved by additionally relying on collision-resistant hash functions to compute verification key digests.

SNARKs for NP are currently only known based on heuristics or non-standard assumptions [Mic94, BCCT13]. Moreover, SNARK constructions with an explicit knowledge extractor are not known and are subject to strong barriers [BCPR16]. We ask:

*Can we realize monotone-policy aggregate signatures from standard assumptions?*

**Signatures from Batch Arguments.** Non-interactive batch arguments (BARGs) [BHK17, KPY20, CJJ21] allow an efficient prover to compute a publicly verifiable proof of the validity of multiple NP statements, with size sublinear in the total witness length. If at least one statement is false, no polynomial-time adversary should be able to compute an accepting proof.

A recent sequence of works [CJJ21, CJJ22, WW22, CGJ+23, KLVW23] provided constructions of BARGs for NP in the common reference string (CRS) model from various standard assumptions including learning with errors (LWE), decisional linear assumption (DLIN) over pairing groups, and sub-exponential decisional Diffie Hellman (DDH). Notably, these works achieve a *somewhere extraction* property that guarantees (given a CRS "trapdoor") efficient extraction of the witness of one statement in the batch from any accepting proof. This guarantee holds in the adaptive setting where the adversary can choose the NP statements as a function of the CRS. Using this property, recent works obtained the first constructions of aggregate signatures from standard assumptions [WW22, DGKV22, CJJ21].

It is easy to see that there is a direct correspondence between the $k$-out-of-$k$ (i.e., conjunction) policies supported by BARGs and aggregate signatures. In light of this connection, and towards answering the above question, we ask whether there exist batch arguments that support more expressive policies of the following form: given a batch of statements $(x_1, \ldots, x_k)$ and a monotone policy $f : \{0,1\}^k \to \{0,1\}$, a valid proof should attest whether $f(b_1, \ldots, b_k) = 1$, where $b_i$ indicates the veracity of $x_i$.

A very recent work [BBK+23] investigates this problem for the setting where the policy $f$ is described by a polynomial-size monotone circuit. They achieve an extraction property that extends the prior notion of somewhere extraction to general monotone policies. However, as we discuss in Section 2, this guarantee turns out to be inadequate for achieving monotone-policy aggregate signatures.

## 1.1 Our Results

We present the first constructions of aggregate signatures from standard assumptions that support expressive monotone policies and achieve poly-logarithmic signature size and verification time. Our constructions are in the common reference string (CRS) model where the size of the CRS grows only poly-logarithmically in the number of signers.

Before we state our results, we elaborate on our notion of aggregate signatures for monotone policies.

**Monotone-Policy Aggregate Signatures.** In an aggregate signature scheme for a monotone policy $f$, each party $\mathsf{P}_i$ publishes its own verification key $\mathsf{vk}_i$, and there is a deterministic public algorithm that aggregates the verification keys of all $k$ parties into a single aggregated verification key $\widehat{\mathsf{vk}}$. Given a collection of signatures $\{\sigma_i\}_{i \in I}$ on a message $m$, an aggregator can produce an aggregated signature $\widehat{\sigma}$ on $m$.

The signature $\widehat{\sigma}$ verifies with respect to $\widehat{\mathsf{vk}}$ if $f(b_1, \ldots, b_k) = 1$, where $b_j = 1$ for every $j$ such that signature $\sigma_j$ verifies with respect to $\mathsf{vk}_j$.

In terms of security, we require that no efficient adversary can win the following game with non-negligible probability. The adversary may ask the challenger for the verification key of any party, and it can ask for a signature on any message under any of these verification keys. These queries can be completely *adaptive*. Finally, the adversary produces a sequence of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$ that may include keys that were not generated by the challenger. Intuitively, we think of the keys generated by the challenger as belonging to honest parties while the other keys come from parties corrupted by the adversary. Together with the verification keys, the adversary also produces a target message $m$ and a signature $\widehat{\sigma}$. The adversary wins if $\widehat{\sigma}$ verifies with respect to $m$ and the key $\widehat{\mathsf{vk}}$ aggregated from $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$, and if the forgery is non-trivial. Intuitively, the forgery is non-trivial if the set of honest parties that signed $m$ together with all corrupted parties does not satisfy the policy. That is, $f(b_1, \ldots, b_k) = 0$ where $b_j = 1$ for every $j$ such that either the adversary asked for signature on $m$ with respect to $\mathsf{vk}_j$, or $\mathsf{vk}_j$ was not generated by the challenger.

**Aggregate Signatures for Bounded-Space Monotone Policies.** Our first construction supports read-once, bounded-space monotone policies. Such policies are given by a monotone function $f : \{0,1\}^k \to \{0,1\}$ that is computable by an algorithm that reads each bit of the input once, and maintains a state of size $S$ that is updated after reading each bit. This class of monotone policies includes, for example: the $t$-out-of-$k$ threshold function with state of size $S = \log k$, or the weighted threshold function for weights in $[B]$ with state of size $S = \log kB$.

**Theorem 1.1** (Informal)**.** *Assuming the existence of somewhere extractable BARGs with $2^S$-security, there exists an aggregate signature scheme for all read-once $O(S)$ space polynomial-time monotone policies. The size of the CRS, the size of the aggregated signature, and the verification time are* $\mathrm{poly}(\log k, S, \lambda)$ *where $\lambda$ denotes the security parameter.*

In particular, for $k = \mathrm{poly}(\lambda)$ and $S = \log(k)$ the theorem only requires BARGs with polynomial security.

Under the same assumptions as in Theorem 1.1, we can also construct aggregate signatures for monotone policies $f$ that are computable by monotone circuits of fan-in two and $\log(\lambda)$-depth (or more generally, $d$-depth under $2^d$-secure BARGs). This result is incomparable to Theorem 1.1. We refer the reader to Appendix A for further details.

**Fast Aggregator.** For threshold policies, our result in Theorem 1.1 can be extended to achieve a more efficient aggregator whose running time only depends on the number of signatures $t$ as opposed to the total number of parties $k$.

**Theorem 1.2** (Informal)**.** *Assuming the existence of somewhere extractable BARGs, there exist aggregate signature schemes for threshold policies such that the size of the CRS, the size of an aggregated signature and the verification time are* $\mathrm{poly}(\log k, \lambda)$*, and the aggregation time is* $\mathrm{poly}(t, \lambda)$ *for threshold $t$.*

**Weakly Unforgeable Aggregate Signatures for Polynomial-Size Monotone Policies.** Our last construction of aggregate signatures supports any policy computable by polynomial-size monotone circuits. It achieves the same (poly-logarithmic) CRS size, signature size and verification time as in Theorem 1.1 but satisfies a weaker notion of security. Intuitively, this notion only guarantees that the adversary cannot forge signatures on messages that were not signed by any honest party. This definition relaxes the fully

adaptive definition discussed above as follows: in the security game, signing queries on the challenge message $m$ under any verification key generated by the challenger are not allowed. (The adversary is still free to output its own verification keys.) A similar relaxation was studied in the context of threshold signatures [Sho00, BTZ22] where the stronger notion of full-security has proven to be more useful, but also more challenging to achieve.[2]

**Theorem 1.3** (Informal)**.** *Assuming fully homomorphic encryption and somewhere extractable BARGs, there exist* weakly unforgeable *aggregate signature schemes for all polynomial-size monotone circuits. The size of the CRS, the size of an aggregated signature and the verification time is* $\mathrm{poly}(\log k, \lambda)$.

**Universal Aggregation and Updatability.** We now highlight some additional properties of our constructions of aggregate signatures. First, our scheme supports *universal aggregation* [HKW15]: we allow each user to use a different signature without modifying the way that individual verification keys or signatures are generated.[3] As a result, a signature issued by a party $i$ can be *reused* for computing aggregate signatures with respect to different aggregate verification keys for different set of parties (that include $i$). Second, our schemes allow new users to *dynamically* join the system without the need to rerun system setup. The aggregate verification key defined for any subset of users can be efficiently updated to add new users. (The aggregated verification key in our scheme is simply a root of a hash tree computed over the individual verification keys.)

**On Multi-hop Aggregation.** Recently, [DGKV22] used rate-1 BARGs to construct multi-hop $n$-out-of-$n$ aggregate signatures, where signatures can be aggregate incrementally. While, our schemes are only presented for the single-hop setting, we follow the same BARG based approach as [DGKV22]. Therefore, it is plausible that similar ideas could be applied to our schemes to support multi-hop aggregation; we leave further exploration of this topic to future work.

**Monotone-Policy Aggregate Signatures from Batch Arguments.** Our aggregate signatures constructions are based on new notions of monotone-policy BARGs. To realize Theorem 1.1 we rely on monotone-policy BARGs with a new notion of security that we called *adaptive subset extraction*. Our definition models an adaptive adversary that given the CRS outputs a batch of $k$ statements, a proof $\pi$ and a necessary subset $J$ for the monotone policy $f$. Here, we say that a subset $J \subset [k]$ is *necessary* for $f$ if for every input $b_1, \ldots, b_k$ such that $f(b_1, \ldots, b_k) = 1$, there exists a $j \in J$ such that $b_j = 1$. Our definition requires an efficient extractor algorithm that extracts from $\pi$ a witness $w$ such that conditioned on $\pi$ being an accepting proof and $J$ being a necessary subset, the extractor outputs a valid witness $w_j$ for an index $j \in J$ with probability at least $\frac{1}{k}$. Since the extractor outputs a witness without knowing the set $J$, hitting $J$ with probability $\frac{1}{k}$ is optimal.[4]

This notion is incomparable to the notion of subset extraction for monotone-policy BARGs from [BBK+23]. In their definition, the necessary subset $J$ is chosen non-adaptively and programmed into the CRS, but the extraction algorithm is required to succeed with probability negligibly close to 1. Our construction of monotone-policy aggregate signatures crucially relies on the adaptive nature of our new security definition.

We construct BARGs with adaptive subset extraction for read-once bounded-space monotone policies.

[2] See single- vs. dual-parameter threshold scheme in [Sho00] and TS-UF-0 vs. TS-UF-1 in [BTZ22].

[3] For Theorem 1.3 the verification keys of the base signature are required to satisfy some natural property. See Section 5.4 for further details.

[4] For example in the case of $k$-out-of-$k$ threshold, each singleton set $\{j\}$ for $j \in [k]$ is necessary.

**Theorem 1.4** (Informal). *Assuming the existence of somewhere extractable* BARG*s with* $2^S$*-security, there exist* BARG*s with adaptive subset extraction for all read-once* $O(S)$ *space polynomial-time monotone policies. The proofs are of size* $|w| \cdot \mathrm{poly}(\log k, S, \lambda)$ *and the CRS is of size* $\mathrm{poly}(\log k, \lambda)$*, where* $\lambda$ *denotes the security parameter and* $|w|$ *is the length of a single witness.*

We also give a variant of the BARGs in Theorem 1.4 where the running time of the prover grows with the number of witnesses it is given instead of the total number of statements $k$. This gives Theorem 1.2.

Finally, we realize Theorem 1.3 based on BARGs for monotone policies with a new notion of security called *functional subset extraction* which generalizes the notion of subset extraction from [BBK+23]. In BARGs with subset extraction, we program a necessary subset $J$ into the CRS. In contrast, in functional subset extraction, we program a function $g$ into the CRS. We also add a tag $y$ that is given as an additional input to the BARG prover and verifier. The requirement is that given any proof $\pi$ that is accepted with respect to a tag $y$, if $J = g(y)$ is a necessary subset then we can extract from $\pi$ a valid witness $w_j$ for an index $j \in J$ with probability negligibly close to 1.

Based on the construction of [BBK+23], we obtain BARGs with functional subset extraction for all policies computable by polynomial-size monotone circuits. The proof is of size $|w| \cdot \mathrm{poly}(\log k, \lambda)$ where $|w|$ is the size of a witness, and the CRS is of size $B \cdot \mathrm{poly}(\log k, \lambda)$, where $B$ is a bound on the description size of a function that can be programmed into the CRS.

## 1.2 Related Work

**Threshold Signatures.** There is an extensive body of work dedicated to the study of threshold signatures [Des88, DF90]. The de facto design of such schemes involves a distributed setup phase to compute a verification key and a threshold secret sharing [Sha79] of the signing key. Subsequently, the parties can jointly sign messages using their key shares, typically using an interactive signing protocol.

While any signature scheme can be "thresholdized" using secure multiparty computation [GMW87, CCD88, BGW88], prior work has primarily focused on designing efficient thresholdizers for popular signature schemes used in practice such as Schnorr [Sch90], ECDSA [CMRR23] and BLS [BLS01]. All such solutions, however, require *interactive* protocols for key setup and (except for BLS) signature computation. Furthermore, by their very design, such solutions do not support universal aggregation of signatures from different signature schemes. Very recently, [GJM+23, DCX+23] overcame the former limitation by presenting an efficient instantiation of the generic SNARK-based approach discussed earlier. The security of their schemes is proven in idealized models.

**Aggregate Signatures.** Aggregate signatures [BGLS03] support aggregation of signatures from different parties on possibly different messages. The key requirement is succinctness of the aggregated signature. Most known schemes also support succinct verification given a digest of the verification key of all the signers. A variation of aggregate signatures where all signers sign the same message is referred to as multisignatures [IN83]. For simplicity of exposition, we use the common terminology of aggregate signatures for both of these settings throughout this work.

Until recently, aggregate signatures were only known in (i) the Random Oracle model from pairing-based assumptions [BGLS03]; or (ii) the standard model utilizing heavy tools such as multilinear maps [FHPS13, RS09] or indistinguishability obfuscation [HKW15], or only achieved weaker properties (e.g., multisignatures or sequential aggregation) [LMRS04, LOS+06]. Recently, using non-interactive BARGs, new constructions in the common reference string (CRS) model were obtained based on a variety of standard

assumptions including LWE [CJJ22, DGKV22, PP22], DLIN assumption over pairing groups [WW22], and sub-exponential DDH [CGJ+23].

**Batch Arguments.** [KPY19] gave the first construction of BARGs for NP from a non-standard but falsifiable assumption over bilinear maps. Recently, a sequence of works [CJJ21, CJJ22, HJKS22, WW22, CGJ+23] constructed BARGs for NP from a variety of standard assumptions with proof sizes ranging from slightly sublinear to poly-logarithmic in the number of statements. Two lines of work devised efficiency boosting compilers for BARGs: [KLVW23] show how to achieve poly-logarithmic proof sizes generically from sublinear-size proofs, and [PP22, DGKV22] construct rate-1 BARGs from BARGs with low rate. Very recently, [BBK+23] constructed BARGs for monotone circuits from polynomial hardness of LWE.

**Sigma Protocols.** Sigma protocols are three-round public-coin zero-knowledge proof systems with special soundness [Cra97]. They have many applications including efficient constructions of digital signatures via the Fiat-Shamir heuristic [FS86]. Sigma protocols are known to be closed under different types of composition such as monotone span programs [CDM00] and more. In the composition of sigma protocols, the communication complexity typically grows with the number of instances and the primary emphasis is on preserving the zero knowledge property. In contrast, our focus is on succinctness and our signatures do not have any hiding property.

## 2 Technical Overview

In this section, we provide an overview of our definitions and constructions. Our starting point is a simple template construction of aggregate signatures from BARGs. By instantiating this template with existing BARG constructions, we get aggregate signatures for the $k$-out-of-$k$ (i.e., conjunction) policy. In what follows, we instantiate the template construction with more general notions of BARGs resulting in aggregate signature for more expressive policies. We start by describing the template construction focusing on the simple case of conjunctions.

**Template Aggregate Signature from BARGs.** The CRS of the aggregate signature scheme consists of a CRS for the BARG and a description of a collision-resistant hash function H. Fix a base signature scheme S that is used by each party to generate keys and sign/verify messages. To aggregate a sequence of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$ we compute a hash tree over them using H and set the aggregate verification key $\widehat{\mathsf{vk}}$ to its root. An aggregate signature $\widehat{\sigma}$ on a message $m$ under $\widehat{\mathsf{vk}}$ is a BARG proof that $m$ has a signature under each of the verification keys. In more detail, let $\mathcal{L}$ be the NP language that contains tuples $(\widehat{\mathsf{vk}}, m, i)$ if and only if there exists a verification key $\mathsf{vk}_i$, a path authenticating $\mathsf{vk}_i$ as the $i$-th leaf of the hash tree rooted at $\widehat{\mathsf{vk}}$, and a valid signature $\sigma_i$ on $m$ under $\mathsf{vk}_i$ using the base scheme S. Given verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$, a message $m$ and signatures $\sigma_1, \ldots, \sigma_k$, we set the aggregate signature $\widehat{\sigma}$ to be a BARG proof for the $k$ statements $(\widehat{\mathsf{vk}}, m, i) \in \mathcal{L}$ for $i \in [k]$. To verify the aggregate signature we simply verify that the BARG proof is accepting.

The succinctness of this construction follows from that of the hash tree and the BARG. The size of the aggregate verification key (i.e. the root of the hash tree) is $\mathrm{poly}(\lambda)$ where $\lambda$ is the security parameter. By the succinctness of the BARG, the size of the aggregate signature (i.e. the BARG proof) is polynomial in $\lambda$ and the size of the witness for a single statement $(\widehat{\mathsf{vk}}, m, i) \in \mathcal{L}$. The witness consists of a key and signature of the base scheme S and an authentication path in the hash tree and it is, therefore, of size $\mathrm{poly}(\lambda, \log k)$.

To get efficient verification we rely on BARGs with efficient verification known as BARGs for the index language [CJJ22]. In such BARGs, verifying the $k$ statements that differ from each other only in the index $i \in [k]$ takes time that is polynomial in the size of just one statement and the proof. Therefore, verifying the $k$ statements $(\widehat{\mathsf{vk}}, m, i) \in \mathcal{L}$ for $i \in [k]$, takes time $\mathrm{poly}(\lambda, \log k)$.

To show the security of the construction, consider an adversary A that outputs verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$, a message $m$, and an aggregate signature $\widehat{\sigma}$ such that $\widehat{\sigma}$ is a valid signature on $m$ under the aggregate key $\widehat{\mathsf{vk}}$ and there exists at least one index $j^* \in [k]$ such that the challenger generated the key $\mathsf{vk}_{j^*}$ and did not sign $m$ under $\mathsf{vk}_{j^*}$. Our goal is to give a reduction that can extract from the aggregate signature $\widehat{\sigma}$ (i.e. the BARG proof) a forged signature on $m$ under $\mathsf{vk}_j$ (or, alternatively, a collision in $H$). This seems to call for BARGs that are not only sound, but also extractable (i.e. BARGs of knowledge). Moreover, the BARGs should be adaptively secure since the statements depend on $\widehat{\mathsf{vk}}, m$ which A may choose as a function of the CRS.

Existing BARG constructions, however, are not known to satisfy full-fledged knowledge soundness in the adaptive setting. Indeed, such BARGs would imply SNARKs for all of NP [BHK17]. Instead, existing BARGs provide a weaker guarantee known as *somewhere extractablilty* [CJJ22]. In a somewhere extractable BARG we can generate, for every index $j \in [k]$, a CRS that is "programmed" at $j$ and is computationally indistinguishable from an honestly generated CRS. Under the programmed CRS, we can extract a witness for the $j$-th statement from any accepting BARG proof using a trapdoor. Using somewhere extractable BARGs, the security reduction for the aggregate signature is as follows. The reduction generates a CRS for the BARG that is programmed at a random index $j \in [k]$. If A produces a valid aggregate signature $\widehat{\sigma}$ (i.e. an accepting BARG proof) the reduction extracts a witness for the $j$-th statement $(\widehat{\mathsf{vk}}, m, j) \in \mathcal{L}$. Such a witness contains a verification key $\widetilde{\mathsf{vk}}_j$, a path authenticating $\widetilde{\mathsf{vk}}_j$ under $\widehat{\mathsf{vk}}$, and a valid signature $\sigma_j$ on $m$ under $\widetilde{\mathsf{vk}}_j$. If $\widetilde{\mathsf{vk}}_j \neq \mathsf{vk}_j$ the reduction finds a collision in H. Otherwise, if $j = j^*$, the reduction finds a forged signature on $m$ under $\mathsf{vk}_{j^*}$. It remains to show that indeed $j = j^*$ with noticeable probability. This follows from the fact that the programmed CRS hides the index $j$. (Note that the reduction finds $j^*$ efficiently and without the CRS trapdoor).

**Towards More Expressive Policies.** Going beyond conjunctions, we turn our attention to aggregate signatures for more expressive monotone policies. Before describing our schemes, we start with a simple construction of an aggregate signature scheme for general monotone policies, albeit, with weak succinctness. The high-level idea is to use the aggregate signature scheme for conjunctions described above and restrict it to the subset of the parties that signed the message. In more detail, given verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$ we again set the aggregate verification key $\widehat{\mathsf{vk}}$ to be the root of the hash tree over the $k$ keys. Let $\{\sigma_i\}_{i \in I}$ be a collection of signatures on a message $m$ by a subset of the parties $I \subseteq [k]$ that satisfies the policy $f$. We aggregate $\{\sigma_i\}_{i \in I}$ into a signature $\widehat{\sigma}$ under $\widehat{\mathsf{vk}}$ as follows. First, we use the conjunction scheme to aggregate $\{\mathsf{vk}_i\}_{i \in I}$ into an key $\widehat{\mathsf{vk}}_I$ and aggregate $\{\sigma_i\}_{i \in I}$ into a signature $\widehat{\sigma}_I$ under $\widehat{\mathsf{vk}}_I$. The aggregate signature $\widehat{\sigma}$ contains contains a description of the set $I$, the keys $\{\mathsf{vk}_i\}_{i \in I}$ together with their authentication paths under $\widehat{\mathsf{vk}}$, and the signature $\widehat{\sigma}_I$. To verify $\widehat{\sigma}$ under $\widehat{\mathsf{vk}}$ we first check that the set $I$ satisfies the policy and that all the authentication paths are valid. Then we use the conjunction scheme to recompute the aggregate key $\widehat{\mathsf{vk}}_I$ and verify the signature $\widehat{\sigma}_I$. The main drawback of this approach is that size of the aggregate signature $\widehat{\sigma}$ can grow with the size of the authorized subset $I$. For example, for the $t$-out-of-$k$ threshold policy, the signature will grow with the threshold $t$. In contrast, our results give succinct aggregate signatures of size $\mathrm{poly}(\log k, \lambda)$ even for policies where the authorized subsets may be much larger.

**Aggregate Signatures from Monotone-Policy BARGs.** Our aggregate signature constructions follow a different approach: starting from the template construction of aggregate signature, we replace the BARGs with the stronger notion of BARGs for monotone policies. Intuitively, a BARG for a monotone policy $f$, can prove that a batch of statements $x_1, \ldots, x_k$ satisfies $f$. That is, $f(b_1, \ldots, b_k) = 1$, where $b_i$ indicates the veracity of $x_i$. Recall that in our template aggregate signature construction, a valid statement $x_i$ corresponds to a signature under the $i$-th party's verification key. BARGs for all policies given by polynomial-size monotone circuits were constructed by a recent work of Brakerski, Brodsky, Kalai, Lombardi and Paneth [BBK+23], however, the security notion guaranteed by their BARGs seems insufficient to prove the security of the aggregate signatures. Instead, our results are based on new monotone-policy BARG constructions that satisfy different security notions. Before expanding on these contributions, we start by reviewing the security notion of [BBK+23] and explain why it is insufficient.

The work of [BBK+23] puts forward a generalization of the somewhere extractability property of [CJJ22] to the settings of monotone-policy BARGs: Instead of programming the CRS on a single index $j \in [k]$, we can program the CRS on any *necessary subset* of indices $J \subseteq [k]$ and the programmed CRS hides $J$. A subset $J \subseteq [k]$ is necessary if any other subset $J'$ that satisfies the policy intersects $J$. Or, equivalently, $J$ is necessary if its complement $[k] \setminus J$ does not satisfy the policy.[5] Using the programmed CRS, we can extract from any accepting BARG proof, a witness for the $j$-th statement for some $j \in J$. This holds even if the adversary can choose the BARG statements $x_1, \ldots, x_k \in \mathcal{L}$ adaptively, as a function of the CRS. For example, for the conjunction policy, every singleton set $\{j\}$ for $j \in [k]$ is necessary, and thus, we recover the original notion of somewhere extractability for BARGs.

We can instantiate the template aggregate signature construction with somewhere extractable monotone-policy BARGs, but we do not know how to argue the security of the resulting scheme.[6] Going back to the aggregate signatures security game, consider an adversary A that is given a CRS, and, after interacting with the challenger, outputs verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$, a target message $m$ and an aggregate signature $\widehat{\sigma}$ on $m$ that verifies under the aggregate key $\widehat{\mathsf{vk}}$. Let $J \subseteq [k]$ be the subset of indices of honest parties that did not sign $m$. That is, $J$ contains every index $j$ such that the challenger generated $\mathsf{vk}_j$ but did not sign $m$ under it. Recall that A's forgery is considered non-trivial if the set of honest parties that signed $m$ together with all corrupted parties does not satisfy the policy. In other words, if A wins the game then $J$ must be a necessary subset. Therefore, if we had programmed the CRS on the set $J$, we could have extracted from $\widehat{\sigma}$ (i.e. the BARG proof) a witness for the $j$-th BARG statement $(\widehat{\mathsf{vk}}, m, j) \in \mathcal{L}$ for some $j \in J$. Such a witness must contain a forged signature $\sigma_j$ on $m$ under $\mathsf{vk}_j$ (or, a collision in the hash H) and, since $j \in J$, this is a non-trivial forgery of the base signature scheme S.

This argument already implies aggregate signatures with static security, where the adversary is required to declare the subset of parties that may sign the challenge message before receiving the CRS. In the adaptive setting, however, the necessary subset $J$ is defined by A's queries and output which may depend the CRS.[7] To get around this hurdle we introduce new notions of security of monotone-policy BARGs where the extraction guarantee holds for necessary subsets that can be chosen *adaptively*, after the CRS is fixed. The BARGs behind Theorems 1.1 and 1.3 are described in Sections 2.1 and 2.2 respectively.

[5] We assume that the policy is not a constant function.

[6] Another issue is that in the construction of [BBK+23] the CRS size grows linearly with $k$, while we are aiming for CRS of size $\mathsf{poly}(\lambda, \log k)$.

[7] Note that if the policy has necessary subsets that are sufficiently large, then guessing the set $J$ may result in security loss that is exponential in $k$. Tolerating such loss would mean losing succinctness.

## 2.1 Aggregate Signatures for Bounded-Space Monotone Policies

In this section we first describe our new notion of monotone-policy BARGs with *adaptive subset extraction* which is the main tool behind Theorem 1.1. Then we overview our construction of monotone-policy BARGs with adaptive subset extraction for the class of read-once, bounded-space monotone policies based on somewhere-extractable BARGs. A monotone policy $f : \{0,1\}^k \to \{0,1\}$ is read-once, space $S$ if there exists a polynomial time Turing machine $\Gamma$ and a pair of states $s_0, s_k \in \{0,1\}^S$ such that $f(b_1, \ldots, b_k) = 1$ if and only if there exist states $s_1, \ldots, s_{k-1} \in \{0,1\}^S$ such that $s_i = \Gamma(s_{i-1}, b_i)$ for every $i \in [k]$.

**Monotone-Policy BARGs with Adaptive Subset Extraction.** In BARGs with adaptive subset extraction the CRS is generated together with an extraction trapdoor. However, in contrast to the notion of somewhere extractability, the CRS is not programmed on any particular subset. Instead, the adversary outputs a description of a subset $J^* \subseteq [k]$ together with the statements $x_1, \ldots, x_k \in \mathcal{L}$ and the proof, after receiving the CRS. Ideally, we would like to require that whenever $J^*$ is necessary and the proof is accepting, we can extract a witness for the $j$-th statement for some $j \in J^*$ using a trapdoor. However, this requirement seems too strong since it implies full-fledged knowledge soundness. (I.e. we can extract witnesses for $i$-th statement for any $i \in I$ for some subset $I$ that satisfies the policy.[8]) Instead, we only require that if the adversary outputs a necessary subset $J^*$ and an accepting proof with some probability $\alpha$, then we can extract a witness for the $j$-th statement for some $j \in J^*$ with probability that is at most negligibly smaller than $\alpha/k$. We note that as long as we only extract a single witness from each proof, losing a factor of $k$ in the extraction probability is inherent. Taking, for example, the conjunction policy, if the adversary outputs a necessary subset $\{j\}$ for a random $j \in [k]$, then the extracted witness fits the $j$-th statement with probability at most $1/k$.[9]

By instantiating the template aggregate signature construction with monotone-policy BARGs that satisfy adaptive subset extraction (instead of somewhere extraction), we can fix the flawed analysis above: Consider an adversary A that wins the aggregate signatures security game with some noticeable probability $\alpha$. As before, we let $J^*$ be the subset that contains every index $j$ such that the challenger generated $\mathsf{vk}_j$ but did not sign $m$ under it. If A wins the game then the aggregate signature $\widehat{\sigma}$ is valid (i.e. the BARG proof is accepting) and the set $J^*$ is necessary. Therefore, we are guaranteed to extract a witness for the $j$-th BARG statement $(\widehat{\mathsf{vk}}, m, j) \in \mathcal{L}$ for some $j \in J^*$ with noticeable probability $\alpha/k - \mathsf{negl}$. Such a witness must contain a non-trivial forged signature $\sigma_j$ under $\mathsf{vk}_j$ (or, a collision in the hash H).

**BARGs with Adaptive Subset Extraction For Low-depth Policies.** Our first construction of BARGs with adaptive subset extraction is based on any somewhere extractable BARG, and it supports policies that are computable by monotone circuits of fan-in two and $\log(\lambda)$-depth (or, more generally, $d$-depth under $2^d$-secure BARGs). The work of [BBK+23] gives a simple construction of BARGs for such policies. We observe that their analysis naturally extends to achieve adaptive subset extraction. For completeness, we describe this construction and its analysis in Appendix A. We note, however, that the main construction from [BBK+23] that supports more expressive policies does *not* guarantee adaptive subset extraction out of the box. Next, we discuss the main challenges in achieving adaptive subset extraction for more expressive policies.

[8] To extract, start from the necessary subset $J^* = [k]$, run the extractor, and obtain a witness for the $j$-th statement for some $j \in J^*$. Remove $j$ from $J^*$ and repeat until $J^*$ is no longer necessary.

[9] We can relax adaptive subset extraction by considering a different bound $\beta \leq 1/k$ on the loss in the extraction probability. The notion remains meaningful for any inverse polynomial $\beta$ and is still sufficient for constructing aggregate signatures.

**On Adaptive Subset Extraction from Somewhere Extraction.** A natural approach for constructing monotone-policy BARGs with adaptive subset extraction would be to start from any somewhere-extractable monotone-policy BARG and transform it into a BARG with adaptive subset extraction for the same policy. Indeed, for the conjunction policy such a transformation exists: simply program the CRS on a necessary subset $\{j\}$ for a random $j \in [k]$. To argue that the resulting BARG satisfies adaptive subset extraction, consider an adversary that outputs a necessary subset $J^*$ and an accepting proof for some batch of statements with probability $\alpha$. The somewhere extraction property guarantees that we can extract a witness for the $j$-th statement from any accepting proof. Moreover, since the CRS hides the index $j$, the probability of extracting a witness for the $j$-th statement for $j \in J^*$ is at least $\alpha/k - \text{negl}$.

This transformation, however, does not seem to extend to other policies. Consider, for example, the $(k-1)$-out-of-$k$ threshold policy where every set with more than one index is necessary. We focus on transformations where the CRS is programmed with a subset $J$ sampled from some distribution over necessary subsets. Let A be an adversary that outputs a random subset $J^*$ of size exactly 2 together with an accepting proof. The somewhere extraction property guarantees that we can extract a witness for the $j$-th statement for some $j \in J$ with probability $1 - \text{negl}$. Now, suppose that whenever $J \setminus J^*$ is not empty, the extracted index $j$ is in $J \setminus J^*$. Indeed, we cannot rule out such an adversary since the extracted index $j$ may depend both on the programmed set $J$ and on A's proof (which may be correlated with $J^*$).[10] Since $J$ is of size at least 2 and $J^*$ is a set of size exactly 2 chosen at random after $J$ is already fixed, the probability that $J \setminus J^*$ is empty is at most $\frac{2}{k(k-1)}$. Therefore, $j \in J^*$ with probability that is noticeably smaller than $1/k$.[11]

**Extracting at an Index Instead of a Subset.** In the above attempt to transform somewhere extraction into adaptive subset extraction, the high level idea was to guess some necessary subset $J$ and program it into the CRS. The problem is that we had no control over the statement in $J$ for which we extract a witness. As a result, we had to guess a subset $J$ that is entirely contained in the subset $J^*$ chosen by the adversary. For some policies, however, this loss in the probability of extraction is too high.

We therefore follow a different approach: instead of guessing a necessary subset $J$, we guess the particular index $j \in [K]$ of the statement $x_j$ we wish to extract a witness for, and program $j$ into the CRS. Indeed, since $j$ is hidden from the adversary, the probability that $j \in J^*$ is at least $1/k - \text{negl}$. However, now we can no longer expect to extract a witness for the $j$-statement from any accepting proof. Unless the subset $\{j\}$ happens to be necessary, the adversary might output statements $x_1, \ldots, x_k \in \mathcal{L}$ where the subset $I \subseteq [k]$ of true statements satisfies the policy, but $x_j \notin \mathcal{L}$ and so extraction must fail. Moreover, we cannot rule out an adversary that outputs such statements with probability 1, even when $j$ is sampled randomly. The issue is that the adversary may choose the statements adaptively, as a function of the CRS, without knowing which of the statements are true.

A possible solution to this problem is to have the adversary prove that it knows the subset $I$ of true statements. Indeed, if the adversary outputs a subset $J^*$ that is necessary, then $J^*$ and $I$ intersect. Therefore, if the adversary knows $I$ (i.e. we can efficiently extract $I$ from the adversary) then we can use the fact that $j$ is hidden to argue that, conditioned on $J^*$ being necessary, $j \in J^* \cap I$ with probability at least $1/k - \text{negl}$. The problem with this solution is without resorting to SNARKs for NP, we do not know how to prove knowledge of $I$ without losing succinctness. Nonetheless, this approach turns out to be useful: instead of requiring the prover to know the subset $I$ of true statements, in our solution we make a weaker requirement.

[10] We can even construct a contrived BARG for which this attack is feasible based on the scheme of [BBK+23].

[11] More generally, if a policy $f$ has exactly $T$ "minimal" necessary subsets (that do not contain smaller necessary subsets) then we can transform any somewhere-extractable BARG for $f$ into a BARG for $f$ with adaptive subset extraction where the loss in extraction probability is bounded by $\beta = 1/T$. However for $\beta > 1/T$ we do not know of such a transformation.

Very roughly, we require that it is possible to simulate a subset $\tilde{I}$ that satisfies the policy and is, in some useful sense, indistinguishable from the actual subset $I$ of true statements chose by the adversary. We show how to enforce this weaker requirement using weaker machinery that can be based on BARGs.

**Verifiable PIR.** The main tool we use to construct monotone-policy BARGs with adaptive subset extraction is verifiable private information retrieval (vPIR) recently introduced by Ben-David, Kalai, and Paneth [BDKP22]. In a plain PIR protocol a server holds a database $D = (r_1, \ldots, r_k)$ and a client can query a row $r_i$ while keeping the index $i$ hidden from the server. Each party sends one message, and the size of the server's answer should be sublinear in $k$. In vPIR, we additionally require the server to use a database that satisfies some predicate $P$. Otherwise, its answer should be rejected by the (public) verification algorithm. This security requirement is formalized via the real-ideal paradigm: for every efficient malicious server A, we require that there exists an efficient simulator Sim such that for every index $i$ the output of the following two experiments are indistinguishable:[12]

**Real:** Query $A$ on index $i$. If the answer verifies output the decrypted row $r$. Otherwise output $\perp$.

**Ideal:** Sim samples a database $D = (r_1, \ldots, r_k)$. If $P(D) = 1$ output $r_i$. Otherwise output $\perp$.

One may consider a stronger secure computation style notion where Sim is required to simulate also the view of A in the real experiment (i.e. the query). However, we do not know how to satisfy this stronger notion. The work of [BDKP22] constructs vPIR for read-once, log-space predicates based on somewhere-extractable BARGs. (More generally, they can handle read-once space $S$ predicates based on $2^S$-secure BARGs.) We say that a predicate $P$ is read-once space $S$ if there exists a polynomial time Turing machine $\Gamma$ and a pair of states $s_0, s_k \in \{0,1\}^S$ such that $P(r_1, \ldots, r_k) = 1$ if and only if there exist states $s_1, \ldots, s_{k-1} \in \{0,1\}^S$ such that $s_i = \Gamma(s_{i-1}, r_i)$ for every $i \in [k]$.

**Verifiable PIR for Policies.** When constructing monotone-policy BARGs for a policy $f$, we use vPIR for particular predicates $P_f$ that evaluates the policy $f$ where the $i$-th input bit to the policy is computed by some local polynomial-time predicate $L$ on the $i$-th database row. Moreover, it would be useful to allow the local predicate to depend on some row-specific instance. That is, given instances $x_1, \ldots, x_k$ and a database $D = (r_1, \ldots, r_k)$, we have $P(D) = f(b_1, \ldots, b_k)$ where $b_i = L(x_i, r_i)$ for every $i \in [k]$. Observe that if $f$ is a read-once space $S$ policy, then $P_f$ is also a read-once space $S$ predicate.

Looking ahead, the instances will correspond to the BARG statements $x_1, \ldots, x_k \in \mathcal{L}$ and, therefore, when defining security, we need to allow the adversary to choose these instances adaptively as a function of CRS. While the analysis of [BDKP22] only supports predicates that are chosen non-adaptively, we extend their construction and show that for predicates of form $P_f$, it satisfies a useful notion of security in the adaptive setting.[13] For every efficient malicious server A we require that there exists an efficient simulator Sim such that for every index $i$ the output of the following two experiments are indistinguishable:

**Real:** Query $A$ on index $i$ and obtain instances $(x_1, \ldots, x_k)$ and an answer $a$. If the answer verifies output $(x_i, r)$ where $r$ is the decryption of $a$. Otherwise output $\perp$.

**Ideal:** Sim samples instances $(\tilde{x}_1, \ldots, \tilde{x}_k)$ and a database $D = (r_1, \ldots, r_k)$. If $P(D) = 1$ output $(\tilde{x}_i, r_i)$. Otherwise output $\perp$.

[12] The actual notion we work with is slightly weaker: for every polynomial $q$ there exists a simulator Sim such that the outputs are $1/q$-indistinguishable.

[13] To get BARGs for the index language, our vPIR construction additionally supports fast verification for instances $x_1, \ldots, x_k$ that differ from each other only on the index $i \in [k]$.

One may consider a stronger notion where Sim is required to simulate all the instances $x_1 \ldots, x_k$ chosen by A in the real experiment instead of just $x_i$. However, we do not know how to satisfy this stronger notion. See Section 6 for further details on the notion and construction of vPIR for policies.

**BARGs with Adaptive Subset Extraction from vPIR: First Attempt.** We start with a simple construction of monotone-policy BARGs from vPIR. (We will eventually need to modify this construction to show adaptive subset extraction.)

To construct BARGs for a policy $f$ we will use vPIR for a related predicate $P_f$ over the database of witnesses: for statements $x_1, \ldots, x_k$, and database $D = (w_1, \ldots, w_k)$, we have that $P(D) = f(b_1, \ldots, b_k)$, where $b_i = 1$ if and only if $w_i$ is a valid witness for the $i$-th statement. The CRS of the BARG is a vPIR query for a random index $j \in [k]$. The prover computes the vPIR answer using the database of witnesses $D = (w_1, \ldots, w_k)$ (if the prover is not given a witness for $x_i$ it sets $w_i = \perp$) and outputs it as the proof. The verifier accepts the proof if the vPIR answer verifies. To extract a witness from the proof we decrypt the vPIR answer.

To explain the difficulty in proving adaptive subset extraction, consider the following (flawed) argument. Let A be an adversary that given a CRS (i.e. a vPIR query), outputs a statement $x_1, \ldots, x_k$, an accepting proof (i.e. a vPIR answer) and a necessary subset $J^*$ with probability $\alpha$. By vPIR security, there is a simulator Sim that samples instances $\tilde{x}_1, \ldots, \tilde{x}_k$ and a database $\tilde{D} = (\tilde{w}_1, \ldots, \tilde{w}_k)$ such that $P_f(\tilde{D}) = 1$ with probability $\alpha - \mathsf{negl}$. Moreover, if $j$ is the random index queried in the CRS, then the output $(\tilde{x}_j, \tilde{w}_j)$ of the ideal experiment is indistinguishable from the output $(x_j, w)$ of the real experiment where $w$ denotes the decrypted vPIR answer (this is conditioned on the outputs being different than $\perp$). If $P_f(\tilde{D}) = 1$ then any necessary subset $J^*$ must contain some index $i \in J^*$ such that $\tilde{w}_i$ is a valid witness for $\tilde{x}_i \in \mathcal{L}$. Since $\tilde{D}$ is sampled independently of $j$ we have that $i = j$ with probability $1/k$. That is, with probability at least $\alpha/k - \mathsf{negl}$, $\tilde{w}_j$ is a valid witness for $\tilde{x}_j \in \mathcal{L}$ and $j \in J^*$. By vPIR security, it follows that also in the real experiment, the extracted value, $w$ is a valid witness for $x_j \in \mathcal{L}$ and $j \in J^*$ with probability at least $\alpha/k - \mathsf{negl}$. The problem is that while this argument holds for any fixed necessary subset $J^*$, it may fail if A chooses $J^*$ adaptively, as a function of the CRS (i.e. the vPIR query). Indeed, vPIR security does not guarantee simulation of both $w$ and the view of A together.

**A Simple Analysis for Threshold Policies.** We start by describing an alternative analysis tailored to the case of threshold policies. The solution for general policies is more complex and we discuss it next. For the $t$-out-of-$k$ threshold policy, if the simulated database $\tilde{D} = (\tilde{w}_1, \ldots, \tilde{w}_k)$ satisfies the predicate, then, since the index $j$ is random and independent of $\tilde{D}$, we have that $\tilde{w}_j$ is a valid witness for $\tilde{x}_j \in \mathcal{L}$ with probability at least $\alpha \cdot t/k$. By vPIR security, it follows that also in the real experiment, $w$ is a valid witness for $x_j \in \mathcal{L}$ with probability at least $\alpha \cdot t/k - \mathsf{negl}$. (Note that we can use vPIR security here since this event does not depend on the set $J^*$ or the view of A.) For the $t$-out-of-$k$ threshold policy, every necessary subset is of size at least $k - t + 1$. Therefore, in the real experiment, since A outputs a necessary subset $J^*$ and since the index $j$ is random and hidden from A, we have that $j \in J^*$ with probability at least $\alpha \cdot (k - t + 1)/k - \mathsf{negl}$. Therefore, by the Union bound, $w$ is a valid witness for $x_j \in \mathcal{L}$ and $j \in J^*$ with probability at least $\alpha/k - \mathsf{negl}$.

**Going Beyond Threshold Policies via Composable vPIR.** The analysis above crucially relies on the symmetric structure of threshold policies and it does not seem to extend beyond that. As suggested by failed attempt above, to support general read-once bounded space polices, it is sufficient to simulate the witness $w$ extracted from the proof together with the subset $J^*$ chosen by A. While we do not know if such simulation is possible, our new simulation strategy will take into account the subset $J^*$. Very roughly, the main idea

behind our simulation is to encode the set $J^*$ as another database, and simulate both databases together using the machinery of vPIR. To this end, we introduce a stronger notion of vPIR that remains secure under composition. Intuitively, in a $t$-composable vPIR protocol for predicates $P^1, \ldots, P^t$, the same query can be answered using $t$ different databases, where the $i$-th database is required to satisfy the predicate $P^i$. To capture this, we modify the real and ideal experiments as follows:

**Real:** Query $A$ on index $i$ and obtain instances $(x_1, \ldots, x_k)$ and $t$ answers $a^1, \ldots, a^t$. If all the answers verify output $(x_i, r^1, \ldots, r^t)$ where $r^j$ is the decryption of $a^j$. Otherwise output $\perp$.

**Ideal:** Sim samples instances $(\tilde{x}_1, \ldots, \tilde{x}_k)$ and $t$ databases $D^1, \ldots, D^t$ where $D^i = (r_1^j, \ldots, r_k^j)$. If $P^j(D^j) = 1$ for every $j \in [t]$ output $(\tilde{x}_i, r_i^1, \ldots, r_i^t)$. Otherwise output $\perp$.

We show that the vPIR construction of [BDKP22] is $t$-composable for any $t = O(1)$. Very roughly, what enables such composition is that evaluating read-once $S$ space predicates over $O(1)$ different databases can be done by a singe read-once $O(S)$ space predicate operating over all the databases simultaneously. See Section 6 for further details on our composition theorem.

**BARGs with Adaptive Subset Extraction from Composable vPIR.** Our final construction of BARGs from vPIR remains unchanged, except that in the analysis we rely on the fact that the vPIR is 2-composable. To argue adaptive subset extraction, consider an adversary A against the BARG that given a CRS (i.e. a vPIR query) outputs statement $x_1, \ldots, x_k$, a proof (i.e. a vPIR answer $a$) and a subset $J^*$. We turn A into a 2-composable vPIR adversary A′ that outputs the instances $x_1, \ldots, x_k$, A's vPIR answer $a$ and an additional answer $a'$ that encodes $J^*$ as follows: Let $D' = (b'_1, \ldots, b'_k)$ be the database such that $b'_i = 1$ if and only if $i \notin J^*$. Let $P'_f$ be the predicate such that $P'_f(D') = 1$ if and only if $f(D') = 0$ (i.e. if and only if $J^*$ is necessary). Using vPIR for the predicate $P'_f$, compute the answer $a'$ to the query in the CRS from the database $D'$.

Say A outputs an accepting proof and a necessary subset $J^*$ with probability $\alpha$. When this occurs, A′ outputs two accepting vPIR answers. Therefore, by 2-composable vPIR security, there is a simulator Sim that samples instances $\tilde{x}_1, \ldots, \tilde{x}_k$ and a pair of databases $\tilde{D} = (\tilde{w}_1, \ldots, \tilde{w}_k)$ and $\tilde{D}' = (\tilde{b}'_1, \ldots, \tilde{b}'_k)$ such that $P_f(\tilde{D}) = 1$ and $P'_f(\tilde{D}') = 1$ with probability $\alpha - \mathrm{negl}$. Moreover, if $j$ is the random index queried in the CRS, then the output $(\tilde{x}_j, \tilde{w}_j, \tilde{b}'_j)$ of the ideal experiment is indistinguishable from the output $(x_j, w, b')$ of the real experiment where $w$ and $b'$ denote the decryption of the answers $a$ and $a'$ respectively (this is conditioned on the outputs being different than $\perp$). If $P_f(\tilde{D}) = 1$ and $P'_f(\tilde{D}') = 1$ then there must exist some index $i \in [k]$ such that $\tilde{w}_i$ is a valid witness for $\tilde{x}_i \in \mathcal{L}$ and $\tilde{b}'_i = 0$. Since $\tilde{D}, \tilde{D}'$ are sampled independently of $j$ we have that $i = j$ with probability $1/k$. That is, with probability at least $\alpha/k - \mathrm{negl}$, $\tilde{w}_j$ is a valid witness for $\tilde{x}_j \in \mathcal{L}$ and $\tilde{b}'_j = 0$. By vPIR security, it follows that also in the real experiment, the extracted value, $w$ is a valid witness for $x_j \in \mathcal{L}$ and $b' = 0$ with probability at least $\alpha/k - \mathrm{negl}$. Finally, by the definition of $a'$ (which was computed honestly) we have that $b' = b'_j$ and, therefore, $w$ is a valid witness for $x_j \in \mathcal{L}$ and $j \in J^*$ with probability at least $\alpha/k - \mathrm{negl}$.

**Fast Prover/Aggregator.** For general read-once bounded-space policies, the number of true statements required to satisfy the policy may be large. Therefore, the running time of the honest prover must, in general, grow with $k$. When constructing aggregate signatures from BARGs this affects the time required to aggregate signatures. However, for specific policies we can hope to do better. For example, for the $t$-out-of-$k$ threshold policy, we construct a BARG with adaptive subset extraction where the prover time grows with $t$ instead of $k$. Plugging this BARG into the template aggregate signature construction gives

Theorem 1.2. Going beyond thresholds, the ideas behind our construction can be extended to give schemes with fast prover/aggregator for other policies as well. We discuss such extensions in Section 7.2.

We modify our construction of BARG from vPIR to use a more efficient encoding of the database of witnesses. In the scheme above, the prover constructs a database with $k$ rows where the witness for the $i$-th statement is stored in the $i$-th row, and the remaining rows contain $\perp$. In the modified scheme, the prover constructs a database $D$ with $t$ rows such that each row contains a witness $w$ and an index $j \in [k]$ such that $w$ is a valid witness for the $j$-th statement and, the sequence of $t$ indices stored in the database is strictly increasing. Indeed, such a database exists if and only if at least $t$ out of the $k$ statements are true. Checking that the database satisfies these two properties can be done by a read-once predicate $P$ using $\log k + 1$ bits of space.[14]

The analysis of the new construction is similar to that above, with some key modifications. We again turn any adversary A against the BARG into a 2-composable vPIR adversary A′. When A outputs a vPIR answer $a$ and a necessary subset $J^*$, A′ outputs an additional answer $a'$ as above, however, $a'$ is computed in a different way: The answer $a'$ is computed from a database $D'$ using vPIR for a predicate $P'$ where $D', P'$ are as follows: Let $\bar{J}$ be a set of size $t-1$ such that $J^* \cup \bar{J} = [k]$ (such a set exists since $J^*$ is necessary). Let $j_1, \ldots, j_{t-1}$ be the indices in $\bar{J}$ in increasing order. For $i \in [t]$, the $i$-th row of $D'$ contains the pair of indices $(j_{i-1}, j_i)$ where at the edges we set $j_0 = 0$ and $j_t = k+1$. We can think of the row $(j_{i-1}, j_i)$ as describing an open interval. The predicate $P'$ checks that the intervals in $D'$ are indeed ordered and cover all indices in $[k]$ except for the $t-1$ end points in $\bar{J}$. (Note that $P'$ is indeed a read-once $\log k + 1$ space predicate.) The key property that enables the analysis to go through is that in the ideal experiment, if the simulator samples a pair of databases $\tilde{D}, \tilde{D}'$ such that $P(\tilde{D}) = 1$ and $P'(\tilde{D}') = 1$ then there must exist some index $i \in [t]$ such that if the $i$-th rows of $\tilde{D}$ and $\tilde{D}'$ are $(\tilde{w}, \tilde{j})$ and $(\tilde{j}_{i-1}, \tilde{j}_i)$ respectively, then $\tilde{w}$ is a valid witness for the $\tilde{j}$-th statement and $\tilde{j}_{i-1} < \tilde{j} < \tilde{j}_i$. Intuitively, the latter guarantees that in the real world, the extracted index $j$ is outside the set $\bar{J}$ (and therefore, inside $J^*$) with sufficiently high probability. See Section 7 for further details.

## 2.2 Weakly Unforgeable Aggregate Signatures for Polynomial-Size Monotone Policies

In this section we overview the proof of Theorem 1.3 for constructing aggregate signatures for all monotone policies given by polynomial-size circuits with weakly unforgeable security. Recall that in the aggregate signatures security game, the adversary outputs verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$, a target message $m$ and a signature $\widehat{\sigma}$. The adversary wins if $\widehat{\sigma}$ is a valid signature on $m$ under the aggregate key and if the forgery is non-trivial. In the case of weakly unforgeable security, the forgery is considered non-trivial if the set of corrupted parties, whose keys where not generated by the challenger, does not satisfy the policy, and if no honest party signed $m$. (This is in contrast to the fully adaptive notion where honest parties are allowed to sign $m$, but the set of honest parties that signed $m$ together with all corrupted parties should not satisfy the policy.)

Our construction is, once again, based on the template aggregate signature construction instantiated with a new notion of monotone-policy BARGs. To motivate this new notion we briefly recall our previous attempts: in a nutshell, our proof strategy was to define a necessary subset of indices $J^*$ based on the adversary's queries and then try to extract a witness for the $j$-th BARG statement for some $j \in J^*$. Since $J^*$ was only fixed at the time the adversary outputs its forgery, we could not program $J^*$ into the CRS. Instead, we relied on the stronger notion of adaptive subset extraction where the subset $J^*$ can be chosen by the

[14] The predicate can store the index of the last row read using $\log k$ bits and use another bit to remember if any of the previously read rows violated the constraints.

adversary, as a function of the CRS. Our construction of BARGs with adaptive subset extraction, however, is limited to read-once bounded space policies.

The main idea behind our construction in Theorem 1.3 is to fix the subset $J^*$ as a function of the BARG statements. This will allow us to target $J^*$ not only during extraction, but already in the construction and verification of the BARG proof. In more detail, in the weakly unforgeable aggregate signatures security game, the subset $J^*$ contains the indices of all honest parties whose verification keys were generated by the challenger. (This is in contrast to the fully adaptive notion where $J^*$ does not contain indices of honest parties that signed $m$.) Therefore, to recover the subset $J^*$ from the BARG statements (i.e. the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$) it is sufficient to distinguish keys generated by the challenger from keys generated by the adversary. To this end, we rely on an underlying signature scheme that supports *trapdoor keys*. In such a signature scheme, it is possible to generate "marked" verification keys that can be recognized using a trapdoor. Without the trapdoor, however, the adversary cannot distinguish the marked keys from honestly generated keys, or generate new marked keys.[15]

To implement this idea, we instantiate the template aggregate signature construction with a new notion of monotone-policy BARGs with *functional subset extraction* that generalise the somewhere extraction property. A construction satisfying this notion is implicit in [BBK+23]. Recall that in BARGs with somewhere extraction, if we program the CRS on a necessary subset $J$, then, using a trapdoor, we can extract a witness for the $j$-th statement for some $j \in J$ from any accepting proof. In functional subset extraction, the subset $J$ depends both on the programmed CRS and on an additional input that is fixed together with the BARG statements. In more detail, the CRS is programmed with a function $g$ and it is indistinguishable from an honestly generated CRS.[16] The BARG prover and verifier take an additional input $y$. The guarantee is that if $J = g(y)$ is a necessary subset, then we can extract a witness for the $j$-th statement for some $j \in J$ from any accepting proof using a trapdoor. We additionally require that the verification time does not grow with the running time of $g$. Moreover, in case the input $y$ is itself long, the verifier only requires a short digest of $y$ and its running time does not grow with $|y|$.

We plug in BARGs with functional subset extraction into the template aggregate signature construction. To aggregate a signature under a sequence of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_k$, we use this sequence as the input $y$ to the BARG prover. We also include a short digest (a hash root) of $y$ in the aggregate verification key, and the BARG verifier uses this digest to verify an aggregate signature. In the analysis, we consider an alternative challenger that gives the adversary marked verification keys, and programs the CRS with a function $g$ that is hard-coded with the trapdoor for the base signature. Given an input $y = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$, $g$ uses the trapdoor to recognize the marked keys and outputs the subset $J$ containing their indices. We note that this analysis does not extend beyond weak unforgeability. In the full-fledged unforgeability game, the adversary can ask the challenger to sign the target message $m$ under some of the marked verification keys. In this case, the indices of these keys should not be included in the subset $J$, or we may extract a trivial forgery. In our weakly unforgeable construction, however, the subset $J$ is fixed only as a function of the verification keys chosen by the adversary, regardless of its signing queries.

Our construction of monotone-policy BARGs with functional subset extraction is based on the somewhere extractable BARGs for monotone policies given by polynomial-size circuits from [BBK+23]. In their construction, the CRS contains the programmed subset $J$ encrypted with a fully homomorphic encryption scheme. Therefore, given an input $y$ and a CRS that contains an encryption of a function $g$, the prover homomorphically evaluates a CRS that is programmed on $g(y)$ and computes its BRAG proof with respect

[15] Starting from any signature scheme, we can construct a scheme with trapdoor keys by adding a random string to each verification key. To mark a key, we replace the random string with a PRF-based MAC.

[16] The length of the CRS grows with an upper bound on the description of $g$.

to the evaluated CRS. The verifier given the input $y$ can recompute the evaluated CRS itself and verify the proof. To allow verification given only the hash of $y$ and to reduce the verification time to be independent of the running time of $g$, we use a RAM SNARK for deterministic computation [CJJ22] to delegate the computation of the evaluated CRS to the prover.

# 3 Preliminaries

**Notations.** We use PPT to denote probabilistic polynomial-time, and denote the set of all positive integers up to $n$ as $[n] := \{1, \ldots, n\}$. For any $x \in \{0,1\}^n$ and any subset $J \subset [n]$ we denote by $x_J = (x_j)_{j \in J}$. For any finite set $S$, $x \leftarrow S$ denotes a uniformly random element $x$ from the set $S$. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element $x$ drawn from the distribution $\mathcal{D}$. We will also use the notation $\mathbb{1}_{[k]\setminus J}$ often to denote a vector in $\{0,1\}^k$ such that the $i$-th position is 0 if and only if $i \in J$.

**The universal language.** Let $\mathcal{L}_U$ be the language of all triplets $(\Gamma, x, y, T)$ such that $\Gamma$ is a description of a Turing machine that on input $x$ outputs $y$ in $T$ steps. We write $(\Gamma, x, T) \in \mathcal{L}_U$ as a shorthand for $(\Gamma, x, 1, T) \in \mathcal{L}_U$, i.e., $\Gamma$ accepts $x$ in $T$ steps.

## 3.1 Digital Signatures

In this section we define (standard) digital signatures.

**Syntax.** A digital signature scheme consists of the following polynomial-time algorithms:

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$**.** This is a probabilistic algorithm that takes as input the security parameter $1^\lambda$. It outputs a signing key sk and a verification key vk.

$\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \sigma$**.** This is a probabilistic algorithm that takes as input the signing key sk and a message $\mathsf{m} \in \{0,1\}^\lambda$. It outputs a signature $\sigma$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{m}, \sigma) \to 0/1$**.** This is a deterministic algorithm that takes as input the verification key vk, a message $\mathsf{m} \in \{0,1\}^\lambda$ and a signature $\sigma$. It outputs a bit (1 to accept, 0 to reject).

**Definition 3.1.** *A digital signature scheme* (KeyGen, Sign, Verify) *is required to satisfy the following properties:*

**Correctness.** *For any* $\lambda \in \mathbb{N}$ *and* $\mathsf{m} \in \{0,1\}^\lambda$*,*

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

**Unforgeability.** *For any admissible poly-size adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[ \mathsf{Verify}(\mathsf{vk}, \mathsf{m}^*, \sigma^*) = 1 \quad : \quad \begin{array}{l} (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \end{array} \right] \leq \mathsf{negl}(\lambda) \,,$$

*where we say that* $\mathcal{A}$ *is admissible if it did not query the* $\mathsf{Sign}(\mathsf{sk}, \cdot)$ *oracle with* $\mathsf{m}^*$*.*

**Remark 3.1** (Message space)**.** We assume that the message space is $\{0,1\}^\lambda$. This is without loss of generality, since we can sign arbitrary messages by first applying a hash function (that satisfies targeted collision resistance), then signing the hashed message [NY89].

## 3.2 Hash Family with Local Opening

In this section we recall the definition of a hash family with local opening [Mer88].[17]

**Syntax.** A hash family (HT) with succinct local opening consists of the following algorithms:

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$. This is a PPT algorithm that takes as input the security parameter $1^\lambda$ in unary and outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{rt}$. This is a deterministic poly-time algorithm that takes as input a hash key hk and an input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, and outputs a hash value rt.

$\mathsf{Open}(\mathsf{hk}, x, j) \to \rho$. This is a deterministic poly-time algorithm that takes as input a hash key hk, an input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, and an index $j \in [N]$, and outputs an opening $\rho$.

$\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, b, \rho) \to 0/1$. This is a deterministic poly-time algorithm that takes as input a hash key hk, a hash value rt, an index $j \in [N]$, a bit $b \in \{0,1\}$ and an opening $\rho$. It outputs a bit (1 to accept, 0 to reject).

**Definition 3.2.** *(Properties of* HT*) A* HT *family* (Gen, Hash, Open, Verify) *is required to satisfy the following properties.*

**Opening completeness.** *For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any $x \in \{0,1\}^N$, and any index $j \in [N]$,*

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, x_j, \rho) = 1 & : & \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ & & \mathsf{rt} = \mathsf{Hash}(\mathsf{hk}, x), \\ & & \rho = \mathsf{Open}(\mathsf{hk}, x, j) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

**Succinctness.** *In the completeness experiment above, we have that* $|\mathsf{hk}|+|\mathsf{rt}|+|\rho| = \mathsf{poly}(\lambda)$.

**Collision resistance w.r.t. opening.** *For any poly-size adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, 0, \rho_0) = 1 & : & \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ \wedge\ \mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, 1, \rho_1) = 1 & & (\mathsf{rt}, j, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] = \mathsf{negl}(\lambda).$$

**Remark 3.2.** We say that a hash family with local opening is $\Lambda$-secure, for $\Lambda = \Lambda(\lambda)$, if the collision resistance w.r.t. opening property holds against any $\mathsf{poly}(\Lambda)$-size adversary (as opposed to $\mathsf{poly}(\lambda)$-size) and the probability that the adversary finds a collision is $\mathsf{negl}(\Lambda)$ (as opposed to $\mathsf{negl}(\lambda)$). We refer to this property as $\Lambda$-collision-resistance w.r.t. opening.

**Theorem 3.3** ( [Mer88]). *Assuming the existence of a collision resistant hash family there exists a hash family with local opening (according to Definition 3.2).*

[17] In what follows we use the notation HT to denote a hash family with local opening, where HT symbolizes a Hash Tree construction. We emphasize that we are not restricted to such a construction, and use this notation only to give the reader an example to have in mind.

### 3.3 Somewhere Extractable Batch Arguments (seBARGs)

A batch argument system BARG for an NP language $\mathcal{L}$ enables proving that $k$ NP statements are true with communication cost that is polylogarithmic in $k$. There are many BARG variants which are known to be existentially equivalent under mild computational assumptions (see, e.g., [CJJ22, KVZ21, KLVW23]). In this work, for simplicity in our constructions, we make use of an argument system for "batch index Turing machine SAT" (BatchTMSAT), defined in [BBK+23].

**Definition 3.4.** *The language* BatchIndexTMSAT *consists of instances of the form $x = (M, z, k, T)$, where:*

- *$M$ is the description of a Turing machine.*
- *$z$ is an input string (to $M$)*
- *$k$ is a batch size, and*
- *$T$ is a running time.*

*An instance $x = (M, z, k, T)$ is in* BatchIndexTMSAT *if for all $1 \le i \le k$, there exists a string $w_i$ such that $M(z, i, w_i)$ accepts within $T$ steps.*

We sometimes use the notation $\mathcal{R}(x, i, w_i)$ to denote the relation with instance $(x, i)$ and corresponding witness $w_i$.

**Syntax.** A (publicly verifiable and non-interactive) somewhere extractable batch argument system seBARG for BatchIndexTMSAT consists of the following polynomial-time algorithms:

$\mathsf{Gen}(1^\lambda, i^*) \to (\mathsf{crs}, \mathsf{td})$. This is a probabilistic algorithm that takes as input a security parameter $1^\lambda$, and an index $i^* \in [2^\lambda]$. It outputs a common reference string crs along with a trapdoor td.

$\mathcal{P}(\mathsf{crs}, M, z, 1^T, w_1, \ldots, w_k) \to \pi$. This is a deterministic algorithm that takes as input a crs, Turing machine $M$, input $z$, runtime $1^T$, and $k$ witnesses $w_1, \ldots, w_k$. It outputs a proof $\pi$.

$\mathcal{V}(\mathsf{crs}, x, \pi) \to 0/1$. This is a deterministic algorithm that takes as input a crs, instance $x = (M, z, k, T)$, and a proof $\pi$. It outputs a bit (1 to accept, 0 to reject).

$\mathsf{Extract}(\mathsf{td}, \pi) \to w$. This is a deterministic algorithm that takes as input a trapdoor td and a proof $\pi$. It outputs a witness $w$.

**Definition 3.5** (seBARG)**.** *A somewhere-extractable batch argument scheme* seBARG $= (\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *for* BatchIndexTMSAT *is required to satisfy the following properties:*

**Completeness.** *For any $\lambda \in \mathbb{N}$, any $k, n, m, T \le 2^\lambda$, any instance $x = (M, z, k, T) \in$ BatchIndexTMSAT with $|M|+|z| = n$, any corresponding witnesses $w_1, \ldots, w_k \in \{0,1\}^m$ and any index $i^* \in [k]$,*

$$\Pr\left[ \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, i^*), \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, M, z, 1^T, w_1, \ldots, w_k) \end{array} \right] = 1.$$

**Efficiency.** *In the completeness experiment above, $|\mathsf{crs}|+|\pi| \le m \cdot \mathsf{poly}(\lambda)$. The running time of the verifier is at most $\mathsf{poly}(|\mathsf{crs}|+|\pi|) + \mathsf{poly}(\lambda) \cdot |x|$.*

**Index hiding.** *For any poly-size adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every pair of indices* $i_0, i_1 \in [2^\lambda]$,

$$\Pr\left[\ \mathcal{A}(\mathsf{crs}) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\}, \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, i_b) \end{array}\ \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

**Somewhere argument of knowledge.** *For any poly-size adversary $\mathcal{A}$ and any polynomials $k(\lambda), T(\lambda)$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any index $i^* \in [k]$ and for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\ (x, i^*, w^*) \notin \mathcal{R} \end{array} \quad : \quad \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, i^*) \\ (M, z, \pi) = \mathcal{A}(\mathsf{crs}) \\ w^* \leftarrow \mathsf{Extract}\,(\mathsf{td}, \pi) \\ x = (M, z, k, T) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

**Remark 3.3.** We say that a seBARG scheme is $\Lambda$-secure, for $\Lambda = \Lambda(\lambda)$, if the index hiding property and the somewhere argument of knowledge property hold w.r.t. a $\mathsf{poly}(\Lambda)$-size adversary (as opposed to a $\mathsf{poly}(\lambda)$-size), and the advantage probability is $\mathsf{negl}(\Lambda)$ (as opposed to $\mathsf{negl}(\lambda)$). We refer to these properties as $\Lambda$-index-hiding and $\Lambda$-somewhere-argument-of-knowledge, respectively.

**Remark 3.4.** Given an seBARG, one can naturally extend the definition of the key generation algorithm Gen to take as input an index *set* $I \subset [k]$, as opposed to a single index. $\mathsf{Gen}(1^\lambda, I)$ will simply run $\mathsf{Gen}(1^\lambda, i)$ for every $i \in I$. The prover algorithm $\mathcal{P}$, given a crs that encodes the $|I|$ indices, will simply generate $|I|$ proofs (one for each crs), and the verifier will check these $|I|$ proofs independently.

**Theorem 3.6** ( [CJJ22, WW22, KLVW23, CGJ+23]). *There exists a* $\Lambda$*-secure* seBARG *for* BatchIndexTMSAT *assuming* $\Lambda$*-hardness of* LWE *or* DLIN.

# 4 Aggregate Signatures for Monotone Policies

In this section we define aggregate signature schemes for monotone policies. We additionally define a weaker notion, and define a special case of aggregate signatures where the prover time depends only on the number of instances for which it has witnesses.

**Syntax.** Let $F = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of monotone policies, such that each $f \in F_\lambda$ is a monotone function $f\colon \{0,1\}^k \to \{0,1\}$. Let $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme. An $F$-aggregation scheme for S consists of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**.** This is a probabilistic setup algorithm that takes as input the security parameter $1^\lambda$. It outputs a common reference string crs.

$\mathsf{KeyAgg}(\mathsf{crs}, \{\mathsf{vk}_i\}_{i\in[k]}) \to \widehat{\mathsf{vk}}$**.** This is a deterministic key aggregation algorithm that takes as input the common reference string crs, and a collection of verification keys $\{\mathsf{vk}_i\}_{i\in[k]}$. It outputs an aggregate verification key $\widehat{\mathsf{vk}}$.

$\mathsf{SigAgg}(\mathsf{crs}, f, \{\mathsf{vk}_i, \sigma_i\}_{i\in[k]}, \mathsf{m}) \to \widehat{\sigma}$**.** This is a deterministic signature aggregation algorithm that takes as input the common reference string crs, a policy $f$, a collection of verification keys and signatures $\{\mathsf{vk}_i, \sigma_i\}_{i\in[k]}$, and a message m. It outputs an aggregate signature $\widehat{\sigma}$.

$\mathsf{AggVerify}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma}) \to 0/1$. This is a deterministic aggregate verification algorithm that takes as input the common reference string crs, a policy $f$, an aggregate verification key $\widehat{\mathsf{vk}}$, a message m, and an aggregate signature $\widehat{\sigma}$. It outputs a bit (1 to accept, 0 to reject).

**Remark 4.1.** For simplicity, in this work we focus on a notion that only allows for aggregating many signatures on the same message. One might consider a more general notion where each party can sign a different message. In this case the verification algorithm AggVerify should take as input a vector $\{\mathsf{m}_i\}_{i\in[k]}$ of messages instead of a single message m.

In general, if we sign different messages, the running time of AggVerify must grow with $k$. The size of the aggregate signature and verification key, however, should remain independent of $k$. Moreover, in case the messages signed have some succinct representation, we may require that the running time of AggVerify grows with the representation size instead of $k$. Indeed, our constructions in subsequent section can be extended to support such succinct verification. See Remark 5.2 for more details.

**Definition 4.1** (Aggregation Scheme)**.** *An $F$-aggregation scheme* (Setup, KeyAgg, SigAgg, AggVerify) *for* S = (KeyGen, Sign, Verify) *is required to satisfy the following properties:*

**Correctness.** *For any $\lambda \in \mathbb{N}$, any policy $f \in F_\lambda$, any message $\mathsf{m} \in \{0,1\}^\lambda$, and any collection of verification keys and signatures $\left\{\mathsf{vk}_i, \sigma_i\right\}_{i\in[k]}$ such that for $b_i = \mathsf{Verify}(\mathsf{vk}_i, \mathsf{m}, \sigma_i)$ it holds that $f(b_1, \ldots, b_k) = 1$,*

$$\Pr\left[\mathsf{AggVerify}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma}) = 1 \quad : \quad \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \widehat{\mathsf{vk}} \leftarrow \mathsf{KeyAgg}(\mathsf{crs}, \left\{\mathsf{vk}_i\right\}_{i\in[k]}) \\ \widehat{\sigma} \leftarrow \mathsf{SigAgg}(\mathsf{crs}, f, \left\{\mathsf{vk}_i, \sigma_i\right\}_{i\in[k]}, \mathsf{m}) \end{array}\right] = 1\ .$$

**Efficiency.** *There exists a fixed polynomial* poly *such that in the correctness experiment above,* $|\widehat{\mathsf{vk}}|+|\widehat{\sigma}|=\mathsf{poly}(\lambda)$.

**Unforgeability.** *For any $\lambda \in \mathbb{N}$ and $f \in F_\lambda$, define the unforgeability game between an adversary $\mathcal{A}$ and a challenger as follows:*

- *The challenger samples* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*, and gives* crs *to $\mathcal{A}$.*
- *The adversary can now make queries to the challenger. Each query is of one of three types:*
  - *Verification key queries: $\mathcal{A}$ can request a verification key from the challenger. The challenger generates* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, gives* vk *to $\mathcal{A}$ and saves the pair* (sk, vk).
  - *Signing queries: Given a verification key* vk *and a message* $\mathsf{m} \in \{0,1\}^\lambda$*, if the challenger previously saved the pair* (sk, vk) *it gives* $\mathsf{Sign}(\mathsf{sk}, \mathsf{m})$ *to $\mathcal{A}$. Otherwise, it gives $\perp$.*
  - *Signing key queries: Given a verification key* vk*, if the challenger previously saved the pair* (sk, vk) *it gives* sk *to $\mathcal{A}$. Otherwise, it gives $\perp$.*
- *At the end of the game the adversary outputs a collection of verification keys $\left\{\mathsf{vk}_i\right\}_{i\in[k]}$, a message $\mathsf{m} \in \{0,1\}^\lambda$, and an aggregate signature $\widehat{\sigma}$.*
- *Let* $\widehat{\mathsf{vk}} = \mathsf{KeyAgg}(\mathsf{crs}, \left\{\mathsf{vk}_i\right\}_{i\in[k]})$*. For $i \in [k]$, let $b_i = 1$ if one of the following conditions holds. Otherwise, let $b_i = 0$:*
  - *$\mathcal{A}$ did not make a verification key query that was answered with* $\mathsf{vk}_i$ *(that is, if* $\mathsf{vk}_i$ *is a maliciously generated key).*
  - *$\mathcal{A}$ made a signing query for* $\mathsf{vk}_i, \mathsf{m}$.

– *$\mathcal{A}$ made a signing key query for* $\mathsf{vk}_i$

- *The adversary $\mathcal{A}$ wins the game if* $f(b_1, \ldots, b_k) = 0$ *and* $\mathsf{AggVerify}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma}) = 1$.

*For any poly-size adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $f \in F_\lambda$, *$\mathcal{A}$ wins the unforgeability game with probability at most* $\mathsf{negl}(\lambda)$.

In the above definition, and throughout the paper we assume that $\mathsf{m} \in \{0,1\}^\lambda$. Our results can be extended using standard techniques to handle arbitrary sized messages $\mathsf{m} \in \{0,1\}^*$ by further assuming collision-resistant hash functions[18].

We also consider a relaxed definition satisfying a weaker notion of unforgeability.

**Definition 4.2** (Weakly Unforgeable Aggregation Scheme)**.** *A weakly unforgeable $F$-aggregation scheme* $(\mathsf{Setup}, \mathsf{KeyAgg}, \mathsf{SigAgg}, \mathsf{AggVerify})$ *for* $\mathsf{S}$ *is required to satisfy correctness and efficiency (as in Definition 4.1), and a weaker notion of unforgeability.*

*The weak unforgeability game is identical to the unforgeability game in Definition 4.1, except that to win the game, in addition to the condition specified in Definition 4.1, the adversary $\mathcal{A}$ may not make any signing query of the form* $(\mathsf{vk}_i, \mathsf{m})$, *or any signing key query* $\mathsf{vk}_i$ *for any* $i \in [k]$.

## 4.1 Fast Aggregation

In the notion of aggregation scheme as defined above, to compute an aggregated signature under an aggregated key $\widehat{\mathsf{vk}}$ the signature aggregator must know all the verification keys aggregated in $\widehat{\mathsf{vk}}$. In what follows we formalize the notion of aggregation scheme with "sublinear" aggregation where the signature aggregator is only given a subset of the verification keys aggregated in $\widehat{\mathsf{vk}}$ and signatures under the keys in this subset such that the subset satisfies the policy. In addition, for every verification key in the subset, the signature aggregator is also given a "proof" that the key is indeed one of the keys aggregated in $\widehat{\mathsf{vk}}$. For a signature aggregator that is given $t$ signatures and verification keys, we want the aggregator run time to depend only on $t$.

**Syntax.** An $F$-aggregation scheme for $\mathsf{S}$ with fast aggregation contains polynomial-time key aggregation and signature aggregation algorithms with the following syntax (the syntax of the setup and aggregated verification algorithm is unchanged):

$\mathsf{KeyAgg}(\mathsf{crs}, \{\mathsf{vk}_i\}_{i\in[k]}) \to (\widehat{\mathsf{vk}}, \{\rho_i\}_{i\in[k]})$**.** This is a deterministic key aggregation algorithm that takes as input the common reference string crs, and a collection of verification keys $\{\mathsf{vk}_i\}_{i\in[k]}$. It outputs an aggregate verification key $\widehat{\mathsf{vk}}$, and openings $\rho_1, \ldots, \rho_k$.

$\mathsf{SigAgg}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \{\mathsf{vk}_i, \rho_i, \sigma_i\}_{i\in[t]}, \mathsf{m}) \to \widehat{\sigma}$**.** This is a deterministic signature aggregation algorithm that takes as input the common reference string crs, a policy $f$, the aggregate verification key $\widehat{\mathsf{vk}}$, a collection of verification keys, openings and signatures $\{\mathsf{vk}_i, \rho_i, \sigma_i\}_{i\in[t]}$, and a message m. It outputs an aggregate signature $\widehat{\sigma}$.

We modify Definition 4.1 to support the syntax above. In particular, the modified correctness property is as follows:

[18] We note that ollision-resistant hash functions were recently shown to be implied by BARGs. [BKP+24]

**Correctness.** For any $\lambda \in \mathbb{N}$, any $k \leq 2^\lambda$, any policy $f \in F_\lambda$, any message $\mathsf{m} \in \{0,1\}^\lambda$, any set $S \subseteq [k]$ such that $f(\mathbb{1}_S) = 1$, and any collection of verification keys and signatures $\{\mathsf{vk}_i, \sigma_i\}$ such that $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{m}, \sigma_i) = 1$ for every $i \in S$,

$$\Pr\left[\mathsf{AggVerify}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma}) = 1 \quad : \quad \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\widehat{\mathsf{vk}}, \{\rho_i\}_{i\in[k]}) \leftarrow \mathsf{KeyAgg}(\mathsf{crs}, \{\mathsf{vk}_i\}_{i\in[k]}) \\ \widehat{\sigma} \leftarrow \mathsf{SigAgg}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \{\mathsf{vk}_i, \rho_i, \sigma_i\}_{i\in S}, \mathsf{m}) \end{array}\right] = 1 \,.$$

# 5 Batch Arguments for Monotone Policies

In this section we provide new definitions for batch arguments, specifically BARGs with adaptive subset extraction, and BARGs with functional subset extraction, both for monotone policies. The definitions extend the definition of BARGs for monotone policies considered in [BBK+23]. We then show how to construct aggregate signatures from BARGs with adaptive subset extraction, and weakly unforgeable aggregate signatures from BARGs with functional subset extraction.

We defer the construction of these BARGs to later technical sections. Our BARGs are going to be defined for the following batch language considered in [BBK+23].

**Definition 5.1.** *The language* MonotonePolicyTMSAT *consists of instances of the form* $x = (f, M, z, T)$*, where:*

- $f\colon \{0,1\}^k \to \{0,1\}$ *is the description of a monotone function.*
- $M$ *is the description of a Turing machine.*
- $z$ *is an input string (to* $M$*), and*
- $T$ *is a running time.*

*An instance* $x = (f, M, z, T)$ *is in* MonotonePolicyTMSAT *if it satisfies any one of the following witness relations. The two relations define the same language, but differ in the witness representation. Notably,* $\mathcal{R}_{\mathsf{sub}}$ *allows for witnesses that are sublinear in the batch size* $k$*, which will allow us to obtain more efficient provers for our batch arguments.*

- *A* $\mathcal{R}_{\mathsf{full}}$*-witness for* $x$ *is* $w = (w_1, \ldots, w_k)$ *such that* $f(b_1, \ldots, b_k) = 1$*, where* $b_i \in \{0,1\}$ *are defined by* $b_i = 1$ *if and only if* $M(z, i, w_i)$ *accepts within* $T$ *steps.*
- *A* $\mathcal{R}_{\mathsf{sub}}$*-witness for* $x$ *is* $w = (i_1, w_1, \ldots, i_t, w_t)$ *such that:*
  1. *For every* $j \in [t]$*, it holds that* $i_j \in [k]$ *and* $M(z, i_j, w_j)$ *accepts within* $T$ *steps.*
  2. $f(b_1, \ldots, b_k) = 1$*, where* $b_i = 1$ *if and only if* $i = i_j$ *for some* $j \in [t]$*.*

Further, let $F = \{F_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of policies, where each $f \in F_\lambda$ is a monotone function $f\colon \{0,1\}^k \to \{0,1\}$. We say that a subset $J \subseteq [k]$ is necessary for a monotone function $f$ if $f(\mathbb{1}_{[k]\setminus J}) = 0$.

## 5.1 Batch Arguments with Adaptive Subset Extraction

The syntax of batch arguments with adaptive subset extraction is similar to the syntax of somewhere-extractable BARGs for monotone policy in [BBK+23], except that the trapdoor generated by the setup algorithm does not depend on a particular subset of indices (or on a single index as in standard somewhere extractable batch arguments [CJJ22]). Instead the adaptive subset extraction guarantee will be defined with respect to necessary subsets chosen by the prover as a function of the crs.

**Syntax.** Let $\mathcal{R}$ be a witness relation for MonotonePolicyTMSAT (either $\mathcal{R}_{\mathsf{full}}$ or $\mathcal{R}_{\mathsf{sub}}$). Let $\alpha = \alpha(\lambda)$ be a polynomial. A $F$-batch argument with $\alpha$-adaptive subset extraction for relation $\mathcal{R}$ consists of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$**.** This is a probabilistic setup algorithm that takes as input the security parameter $1^\lambda$. It outputs a common reference string crs and a trapdoor td.

$\mathcal{P}(\mathsf{crs}, f, M, z, 1^T, w) \to \pi$**.** This is a deterministic prover algorithm that takes as input the common reference string crs, policy $f$, Turing machine $M$, input $z$, runtime $1^T$, and witness $w$ (for the relation $\mathcal{R}$). It outputs a proof $\pi$.

$\mathcal{V}(\mathsf{crs}, x, \pi) \to 0/1$**.** This is a deterministic verification algorithm that takes as input the common reference string crs, instance $x = (f, M, z, T)$, and a proof $\pi$. It outputs a bit (1 to accept, 0 to reject).

$\mathsf{Extract}(\mathsf{td}, \pi) \to (i, w_i)$**.** This is a deterministic extraction algorithm that takes as input the trapdoor td, and a proof $\pi$. It outputs an index $i \in [k]$, and a witness $w_i$.

**Definition 5.2** (Batch Argument for Policies with Adaptive Subset Extraction)**.** *A $F$-batch argument with $\alpha$-adaptive subset extraction* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *for relation $\mathcal{R}$ is required to satisfy the following properties:*

**Completeness.** *For any $\lambda \in \mathbb{N}$, any $n, m, T \le 2^\lambda$, any instance $x = (f, M, z, T) \in$* MonotonePolicyTMSAT *with $|M|+|z| = n$ and corresponding witness $w$ such that $(x, w) \in \mathcal{R}$ and $|w_i| = m$,*

$$\Pr\left[ \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, f, M, z, 1^T, w) \end{array} \right] = 1 \, .$$

**Succinctness.** *In the completeness experiment above, $|\pi| \le m \cdot \mathsf{poly}(\lambda)$. The running time of the verifier is at most $\mathsf{poly}(|\mathsf{crs}|+|\pi|) + \mathsf{poly}(\lambda) \cdot |x|$.*

**$\alpha$-Adaptive Subset Extraction.** *For any polynomial $T(\lambda)$, any poly-size cheating prover $\mathcal{P}^*$, and any sequence $\{f_\lambda \in F_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\Pr_{\mathsf{EXP}}\left[ \; i \in J \;\wedge\; M(z, i, w_i) = 1 \; \right] \ge \frac{1}{\alpha(\lambda)} \cdot \Pr_{\mathsf{EXP}}\left[ \begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array} \right] - \mathsf{negl}(\lambda) \, ,$$

*where* EXP *is the experiment defined as follows:*

- *Generate* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- *Run the cheating prover and obtain* $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
- *Let* $x = (f_\lambda, M, z, T)$.
- *Extract* $(i, w_i) \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)$.

## 5.2 Batch Arguments with Functional Subset Extraction

We now consider a related notion of functional subset extraction, which generalizes the notion of somewhere extractable BARGs for monotone policy defined in [BBK+23]. Here, we program the crs on some function $g$ whose outputs are subsets of $[k]$. The extraction guarantee we will require that if prover and verifier both use a function input $y$, we can extract from the proof a witness from the set $J = g(y)$, as long as $J$ is a necessary subset. Unlike the definition of adaptive subset extraction considered in Section 5.1, where the crs does not require additional input, we note that here we do have to program a crs but the adversary is allowed to adaptively pick the input $y$ to the function $g$.

**Syntax.** Let $\mathcal{R}$ be a witness relation for MonotonePolicyTMSAT (defined in Definition 5.1). The syntax remains largely the same as in the case of BARGs adaptive subset extraction, with the primary difference pertaining to the additional function $g$ specified during setup, and the additional input $y$ used by both parties.

A $F$-batch argument with functional subset extraction consists of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, g) \to (\mathsf{crs}, \mathsf{td})$**.** This is a probabilistic setup algorithm that takes as input the security parameter $1^\lambda$ and a function $g \in G_\lambda$. It outputs a common reference string crs and a trapdoor td.

$\mathcal{P}(\mathsf{crs}, y, f, M, z, 1^T, w) \to \pi$**.** This is a deterministic prover algorithm that takes as input the common reference string crs, function input $y$, policy $f$, Turing machine $M$, input $z$, runtime $1^T$, and witness $w$ (for the relation $\mathcal{R}$). It outputs a proof $\pi$.

$\mathcal{V}(\mathsf{crs}, y, x, \pi) \to 0/1$**.** This is a deterministic verification algorithm that takes as input the common reference string crs, function input $y$, instance $x = (f, M, z, T)$, and a proof $\pi$. It outputs a bit (1 to accept, 0 to reject).

$\mathsf{Extract}(\mathsf{td}, y, \pi) \to (i, w_i)$**.** This is a deterministic extraction algorithm that takes as input the trapdoor td, and a proof $\pi$. It outputs an index $i \in [k]$, and a witness $w_i$.

**Definition 5.3** (Batch Argument for Policies with Functional Subset Extraction)**.** *A $F$-batch argument with $G$-functional subset extraction* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *for relation $\mathcal{R}$ is required to satisfy the following properties:*

**Completeness.** *For any $\lambda \in \mathbb{N}$, any $k, n, m, \ell, T \le 2^\lambda$, any instance $x = (f, M, z, T) \in$* MonotonePolicyTMSAT *with $|M|+|z| = n$ and corresponding witness $w$ such that $(x, w) \in \mathcal{R}$ and $|w_i| = m$, any function $g \in G_\lambda$, and any function input $y \in \{0,1\}^\ell$,*

$$\Pr\left[\ \mathcal{V}(\mathsf{crs}_\mathcal{V}, y, x, \pi) = 1 \quad : \quad \begin{array}{l}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, g)\\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, y, f, M, z, 1^T, w)\end{array}\ \right] = 1\ .$$

**Succinctness.** *In the completeness experiment above, $|\pi| \le m \cdot \mathsf{poly}(\lambda)$ and $|\mathsf{crs}| \le (m + |g|) \cdot \mathsf{poly}(\lambda)$ where $|g|$ is the description length of the function $g$.*

**Key Indistinguishability.** *For any poly-size adversary $\mathcal{A}$, any polynomial $T(\lambda)$ and any functions $g_0, g_1 \in G_\lambda$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\ \mathcal{A}(\mathsf{crs}) = b \quad : \quad \begin{array}{l}b \leftarrow \{0,1\},\\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, g_b)\end{array}\ \right] \le \frac{1}{2} + \mathsf{negl}(\lambda).$$

**$G$-Functional Subset Extraction.** *For any polynomial $T(\lambda)$ and any poly-size cheating prover $\mathcal{P}^*$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$ and $g \in G_\lambda$,*

$$\Pr_{\mathsf{EXP}}\left[\ \begin{array}{l}\mathcal{V}(\mathsf{crs}_\mathcal{V}, y, x, \pi) = 1 \wedge\\ f(\mathbb{1}_{[k]\setminus J}) = 0 \wedge\\ (i \notin J \ \vee\ M(z, i, w_i) = 0)\end{array}\ \right] \le \mathsf{negl}(\lambda).$$

*where* EXP *is the experiment defined as follows:*

- *Generate* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, g)$.
- *Run the cheating prover and obtain* $(M, z, \pi, y) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
- *Let* $x = (f_\lambda, M, z, T)$ *and* $J = g(y)$.
- *Extract* $(i, w_i) \leftarrow \mathsf{Extract}(\mathsf{td}, y, \pi)$.

**Remark 5.1** (Hashed inputs)**.** We additionally support a syntax where the verifier receives a hash of the function input $y$ (for some hash family with local opening), rather than the full input. This allows for an efficient verifier in case the function input is long. Looking ahead, this will be the case in our construction in Section 5.4.

With this syntax, security is modified so that it holds if the verifier receives an honestly computed hash of $y$. We note that this notion can be obtained generically, using RAM SNARGs (where the BARG proof contains a RAM SNARG proof that the verifier accepts). A similar transformation was implemented in [BBK+23] Section 5.3.3, to obtain a predicate-extractable hash family with efficient verification.

## 5.3 From Adaptive Subset Extraction to Aggregate Signatures

We present our first transformation, where we construct aggregate signatures from BARGs with adaptive subset extraction, and any signature scheme. Specifically, let $F = \{F_\lambda\}$ be a family of monotone policies. We construct an $F$-aggregation scheme for $\mathsf{S}$ (Definition 4.1) from the following building blocks:

- A digital signature scheme $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$.
- An $F$-batch argument with $\alpha$-adaptive subset extraction (Definition 5.2)

  $$(\mathsf{Setup}_{\mathsf{BARG}}, \mathcal{P}_{\mathsf{BARG}}, \mathcal{V}_{\mathsf{BARG}}, \mathsf{Extract}_{\mathsf{BARG}})$$

  for relation $\mathcal{R}$, where $\alpha$ is some polynomial. For simplicity we assume $\mathcal{R} = \mathcal{R}_{\mathsf{full}}$. We note that a similar construction with $\mathcal{R} = \mathcal{R}_{\mathsf{sub}}$ gives a $F$-aggregation scheme with sublinear aggregation (Section 4.1).
- A hash family with local opening (Definition 3.2)

  $$(\mathsf{Gen}_{\mathsf{HT}}, \mathsf{Hash}_{\mathsf{HT}}, \mathsf{Open}_{\mathsf{HT}}, \mathsf{Verify}_{\mathsf{HT}}) .$$

We describe the aggregate signature algorithms in Fig. 1, and prove the following theorem.

**Theorem 5.4.** *Assuming the existence of a digital signature scheme, hash family and $F$-batch argument with $\alpha$-adaptive subset extraction, the construction given in Fig. 1 is an $F$-aggregation scheme for* $\mathsf{S}$.

**Remark 5.2.** The proof of Theorem 5.4 can be extended to extended to give a stronger notion of aggregated signatures where each party can sign a different message (see Remark 4.1). Moreover, if the the verification algorithm is given as input a succinct representation of the messages signed, its running time grows with the representation size instead of $k$. In more detail, we can support a representation of the messages $\{\mathsf{m}_i\}_{i\in[k]}$, either as a Turing machine that given input $i \in [k]$ outputs $\mathsf{m}_i$ in time $\mathsf{poly}(\lambda)$, or as the root of hash tree over the messages.

If the messages $\{\mathsf{m}_i\}_{i\in[k]}$ are represented by a Turing machine $R$ that generates them, we modify the construction given in Fig. 1 as follows: The machine $M$ takes as input $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, R)$ instead of $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$ and it computes $\mathsf{m} = R(j)$. If the messages $\{\mathsf{m}_i\}_{i\in[k]}$ are represented by a hash root $\hat{\mathsf{m}}$ we modify the construction as follows: The machine $M$ takes as input $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \hat{\mathsf{m}})$ instead of $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$, we add the message $\mathsf{m}_i$ and its authentication path $\rho_j^{\mathsf{m}}$ to the witness $w_j$, and $M$ also checks that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}, \hat{\mathsf{m}}, j, \mathsf{m}_j, \rho_j^{\mathsf{m}}) = 1$.

$\mathsf{Setup}(1^\lambda)$:

1. Generate $\mathsf{hk} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$.
2. Generate $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Setup}_{\mathsf{BARG}}(1^\lambda)$.
3. Output $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.

$\mathsf{KeyAgg}(\mathsf{crs}, \{\mathsf{vk}_i\}_{i\in[k]})$:

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.
2. Output $\widehat{\mathsf{vk}} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$.

$\mathsf{SigAgg}(\mathsf{crs}, f, \{\mathsf{vk}_i, \sigma_i\}_{i\in[k]}, \mathsf{m})$:

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.
2. Compute $\widehat{\mathsf{vk}} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$.
3. Let $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$, and define the Turing machine $M(z, j, w_j)$ which operates as follows:
   - Parse $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$.
   - Parse $w_j = (\rho_j, \mathsf{vk}_j, \sigma_j)$.
   - Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}, \widehat{\mathsf{vk}}, j, \mathsf{vk}_j, \rho_j) = 1$.
   - Check that $\mathsf{Verify}(\mathsf{vk}_j, \mathsf{m}, \sigma_j) = 1$.

   Let $T = \mathsf{poly}(\lambda)$ so that the above pseudocode terminates.
4. For every $j \in [k]$, construct a witness $w_j$ such that $M(z, j, w_j) = 1$, using the $\mathsf{Open}_{\mathsf{HT}}$ algorithm to produce the appropriate openings. Let $w = (w_1, \ldots, w_k)$.
5. Output $\widehat{\sigma} = \mathcal{P}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, f, M, z, 1^T, w)$.

$\mathsf{AggVerify}(\mathsf{crs}, f, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma})$:

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.
2. Define $M, T$ as above and $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$, and let $x = (f, M, z, T)$.
3. Output $\mathcal{V}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, x, \widehat{\sigma})$.

Figure 1: Aggregate Signatures for Bounded Space Monotone Policies $f$ from BARGs with Adaptive Subset Extraction for $f$

**Proof of Theorem 5.4.**

**Correctness.** Follows directly from the completeness and correctness properties of the underlying batch argument, digital signature scheme and hash family with local opening.

**Efficiency.** We have that $|\widehat{\mathsf{vk}}| = \mathsf{poly}(\lambda)$ by the succinctness property of the hash family with local opening, and $|\widehat{\sigma}| = \mathsf{poly}(\lambda)$ by the efficiency of the batch argument with adaptive subset extraction (each witness $w_j$ is of length $\mathsf{poly}(\lambda)$ by efficiency of the hash family and digital signature scheme).

**Unforgeability.** Fix any poly-size adversary $\mathcal{A}$. Assume towards contradiction that there exists a polynomial poly such that for infinitely many $\lambda \in \mathbb{N}$, there exists $f \in F_\lambda$ such that $\mathcal{A}$ wins the unforgeability game with probability $> \frac{1}{\mathsf{poly}(\lambda)}$.

We construct an adversary $\mathcal{B}$ that breaks the unforgeability of the underlying digital signature scheme. Given the security parameter $1^\lambda$, a verification key $\mathsf{vk}^*$, and oracle access to $\mathsf{Sign}(\mathsf{sk}^*, \cdot)$, $\mathcal{B}$ does the following:

1. Let $Q(\lambda)$ be a polynomial upper bound on the number of verification key queries that $\mathcal{A}$ makes. Sample $i^* \leftarrow [Q]$.
2. Simulate the aggregate signature unforgeability game against $\mathcal{A}$.
   - Emulate $\mathsf{Setup}(1^\lambda)$, provide crs to $\mathcal{A}$ while holding on to $\mathsf{td}_{\mathsf{BARG}}$.
   - Whenever $\mathcal{A}$ makes a verification key query, if it is the $i^*$th query, give it $\mathsf{vk}^*$. Otherwise, sample and save $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and give $\mathsf{vk}$ to $\mathcal{A}$.
   - Whenever $\mathcal{A}$ makes a signing query for $(\mathsf{vk}, \mathsf{m})$, if $\mathsf{vk} = \mathsf{vk}^*$, query the signing oracle and return $\sigma = \mathsf{Sign}(\mathsf{sk}^*, \mathsf{m})$. Otherwise, if a pair $(\mathsf{sk}, \mathsf{vk})$ was saved, compute and return $\sigma = \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$. Otherwise, give $\perp$ to $\mathcal{A}$.
   - Whenever $\mathcal{A}$ makes a signing key query for $\mathsf{vk}$, if $\mathsf{vk} = \mathsf{vk}^*$ abort. If a pair $(\mathsf{sk}, \mathsf{vk})$ was saved, return $\mathsf{sk}$. Otherwise, give $\perp$ to $\mathcal{A}$.
3. At the end of the game, let $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \mathsf{m}, \widehat{\sigma}$ be $\mathcal{A}$'s output.
4. Extract $(i, w_i) \leftarrow \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}_{\mathsf{BARG}}, \widehat{\sigma})$.
5. Parse $w_i = (\rho_i, \mathsf{vk}'_i, \sigma_i)$ and output $\mathsf{m}, \sigma_i$.

We also consider an inefficient adversary $\mathcal{B}'$ that is defined exactly like $\mathcal{B}'$, except that if $\mathcal{A}$ makes a signing key query for $\mathsf{vk}^*$, $\mathcal{B}'$ does not abort. Instead, $\mathcal{B}'$ samples a secret key $\mathsf{sk}^*$ such that $(\mathsf{sk}^*, \mathsf{vk}^*)$ are distributed like an output of KeyGen, and reruns $\mathsf{sk}^*$ to $\mathcal{A}$.

Recall that $\alpha$ is the $F$-batch argument's extraction loss. We prove the following claim.

**Claim 5.5.** *In the experiment of running $\mathcal{B}'$, considering the extracted $i, w_i = (\rho_i, \mathsf{vk}'_i, \sigma_i)$, let* GOOD *be the event that:*

- $\mathsf{vk}'_i$ *is a key that $\mathcal{A}$ received via a verification key query.*
- $\mathsf{vk}'_i = \mathsf{vk}_i$ *where* $\mathsf{vk}_i$ *is the $i$-th key in the tuple output by $\mathcal{A}$.*
- *$\mathcal{A}$ did not make a signing query for* $\mathsf{vk}'_i$ *and* $\mathsf{m}$.

- *$\mathcal{A}$ did not make a signing key query for* $\mathsf{vk}'_i$.
- $\mathsf{Verify}(\mathsf{vk}'_i, \mathsf{m}, \sigma_i) = 1$.

*Then, for infinitely many* $\lambda \in \mathbb{N}$, *we have* $\Pr[\text{GOOD}] > \frac{1}{\alpha(\lambda)\cdot \mathrm{poly}(\lambda)}$ .

Before proving Claim 5.5, we first argue that it implies that $\mathcal{B}$ succeeds in breaking unforgeability, in contradiction to the security of the underlying signature scheme S.

Conditioned on GOOD, since $\mathsf{vk}'_i$ was received via a verification key query and $i^*$ is uniform and independent, with probability $\frac{1}{Q(\lambda)}$ we have that $\mathsf{vk}'_i$ was received in the $i^*$th query. Thus, for infinitely many $\lambda$,

$$\Pr_{\mathcal{B}'}\left[\ \mathsf{vk}'_i = \mathsf{vk}^* \ \wedge\ \text{GOOD}\ \right] = \frac{1}{Q(\lambda)} \cdot \Pr_{\mathcal{B}'}\left[\ \text{GOOD}\ \right] > \frac{1}{Q(\lambda)\alpha(\lambda)\mathrm{poly}(\lambda)}\,,$$

In the experiment of running $\mathcal{B}$, let GOOD′ be the event that $\mathcal{B}$ does not abort, $\mathsf{vk}'_i = \mathsf{vk}^*$ and GOOD holds. Since GOOD implies that $\mathcal{A}$ did not make a signing key query for $\mathsf{vk}'_i$ we have that conditioned on $\mathsf{vk}'_i = \mathsf{vk}^*$ and GOOD the executions of $\mathcal{B}$ and $\mathcal{B}'$ are identical. Therefore,

$$\Pr_{\mathcal{B}}\left[\ \text{GOOD}'\ \right] \geq \Pr_{\mathcal{B}'}\left[\ \mathsf{vk}'_i = \mathsf{vk}^* \ \wedge\ \text{GOOD}\ \right] > \frac{1}{Q(\lambda)\alpha(\lambda)\mathrm{poly}(\lambda)}\,,$$

Conditioned on GOOD′, $\mathcal{B}$ outputs $\mathsf{m}, \sigma_i$ such that $\mathsf{Verify}(\mathsf{vk}^*, \mathsf{m}, \sigma_i) = 1$ and it did not make a signing query for $\mathsf{vk}'_i = \mathsf{vk}^*$ and $\mathsf{m}$. This implies that $\mathcal{B}$ succeeds in forging a signature with at least the same non-negligible probability.

**Proof of Claim 5.5.** The claim follows from the $\alpha$-adaptive subset extraction of the underlying batch argument, and the collision resistance with respect to opening of the hash family. We consider a cheating prover $\mathcal{P}^*$, that given $\mathsf{crs}_{\mathsf{BARG}}$, does the following:

1. Simulate the aggregate signature unforgeability game against $\mathcal{A}$, as follows:
   (a) Sample $\mathsf{hk} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$.
   (b) Give $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$ to $\mathcal{A}$.
   (c) Answer the verification key and signing queries made by $\mathcal{A}$, by applying KeyGen and Sign as required.
2. At the end of the game, let $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \mathsf{m}, \widehat{\sigma}$ be $\mathcal{A}$'s output.
3. Let $J \subseteq [k]$ be the set of $i \in [k]$ such that $\mathsf{vk}_i$ is a key that $\mathcal{A}$ received via a verification key query, and $\mathcal{A}$ did not make a signing query for $\mathsf{vk}_i$ and $\mathsf{m}$, or a signing key query for $\mathsf{vk}_i$.
4. Let $M$ as in SigAgg and $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$ where $\widehat{\mathsf{vk}} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$.
5. Output $M, z, \widehat{\sigma}, J$.

Let EXP be the experiment defined in the adaptive subset extraction requirement:

- Generate $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Setup}_{\mathsf{BARG}}(1^\lambda)$.
- Run the cheating prover above and obtain $(M, z, \widehat{\sigma}, J) \leftarrow \mathcal{P}^*(\mathsf{crs}_{\mathsf{BARG}})$.

- Let $x = (f, M, z, T)$.
- Extract $(i, w_i) \leftarrow \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}_{\mathsf{BARG}}, \widehat{\sigma})$.

By assumption, since $\mathcal{A}$ wins the aggregate unforgeability game with probability $> \frac{1}{\mathsf{poly}(\lambda)}$, we have

$$\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, x, \widehat{\sigma}) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] > \frac{1}{\mathsf{poly}(\lambda)},$$

where (i) $\mathcal{V}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, x, \widehat{\sigma}) = 1$ follows from the fact that $\mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma}) = 1$; and (ii) $f(\mathbb{1}_{[k]\setminus J}) = 0$ follows from our definition of $J$. Therefore, by $\alpha$-adaptive subset extraction,

$$\Pr_{\mathsf{EXP}}\left[\ i \in J \ \wedge\ M(z, i, w_i) = 1\ \right] > \frac{1}{\alpha(\lambda)\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)\,.$$

Now, we observe that the experiment of running $\mathcal{B}'$ is identical to EXP, and that the event $i \in J \ \wedge\ M(z, i, w_i) = 1$ above implies GOOD with all but negligible probability, since

- $M(z, i, w_i) = 1$ implies that, parsing $w_i = (\rho_i, \mathsf{vk}'_i, \sigma_i)$, we have that $\rho_i$ is a valid opening of $\widehat{\mathsf{vk}}$ to $\mathsf{vk}'_i$ at location $i$ and $\mathsf{Verify}(\mathsf{vk}'_i, \mathsf{m}, \sigma_i) = 1$. Since $\widehat{\mathsf{vk}}$ is an honestly generated hash, we must have $\mathsf{vk}_i = \mathsf{vk}'_i$ except with negligible probability. This follows from the collision resistance with respect to opening of the hash family, since if $\mathsf{vk}'_i \neq \mathsf{vk}_i$, we would have two distinct openings at the $i$-th location: the honestly computed opening for $\mathsf{vk}_i$ in $\widehat{\mathsf{vk}}$ (can be computed since $\widehat{\mathsf{vk}}$ is honestly computed) and the extracted opening $\rho_i$ to $\mathsf{vk}'_i$.
- $i \in J$ implies that $\mathsf{vk}_i = \mathsf{vk}'_i$ is a key that $\mathcal{A}$ received via a verification key query and $\mathcal{A}$ did not make a signing query for $\mathsf{vk}_i = \mathsf{vk}'_i$ and $\mathsf{m}$.

Therefore, we get that $\Pr[\mathrm{GOOD}] > \frac{1}{2\alpha(\lambda)\cdot\mathsf{poly}(\lambda)}$ for infinitely many $\lambda$, which concludes the proof of Claim 5.5 and Theorem 5.4.

## 5.4 From Functional Subset Extraction to Weakly Unforgeable Aggregate Signatures

We present our second transformation, where we construct weakly unforgeable aggregate signatures from BARGs with functional subset extraction. Our construction will require signature schemes with an additional property that we call *signature scheme with trapdoor keys*. We start by describing this notion, before we move to the construction of the aggregate signature scheme.

### 5.4.1 Signature with Trapdoor Keys

We extend the definition of digital signatures $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ with an additional property of *trapdoor verification keys*. This is a requirement on the key generation algorithm of a signature scheme that when provided with a randomly sampled trapdoor $\mathsf{td} \leftarrow \{0,1\}^\lambda$, we can generate "trapdoor keys". Such trapdoor keys are indistinguishable from honest keys, but their verification keys $\mathsf{vk}$ can be recognized using the corresponding trapdoor. Additionally, without the trapdoor, it is hard to forge new "false positive" verification keys $\mathsf{vk}$ that are recognized as trapdoor keys, even if given a polynomial number of trapdoor key samples.

**Definition 5.6** (Signature with Trapdoor Keys)**.** *A digital signature scheme* $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *has trapdoor keys if it additionally supports the following syntax:*

$\mathsf{KeyGen}(1^\lambda, \mathsf{td}) \to (\mathsf{sk}, \mathsf{vk})$. *This is a* PPT *algorithm that takes as input the security parameter* $1^\lambda$ *and a trapdoor* $\mathsf{td} \in \{0,1\}^\lambda$. *It outputs a signing key* sk *and a verification key* vk.

$\mathsf{Extract}(\mathsf{td}, \mathsf{vk}) \to \{0,1\}$. *This is a deterministic polynomial-time algorithm that takes as input a trapdoor* $\mathsf{td} \in \{0,1\}^\lambda$ *and a verification key* vk. *It outputs a bit (*0 *if it is a normal key,* 1 *if it is a trapdoor key).*

*We additionally require the following properties:*

**Key Indistinguishability.** *For any poly-size adversary* $\mathcal{A}$*, and for any polynomial* $n(\lambda)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathcal{A}\left(\left\{\mathsf{sk}_i^b, \mathsf{vk}_i^b\right\}_{i\in[n]}\right) = b \quad : \quad \begin{array}{l} \mathsf{td} \leftarrow \{0,1\}^\lambda \\ \forall i \in [n],\ \left(\mathsf{sk}_i^0, \mathsf{vk}_i^0\right) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{td}) \\ \forall i \in [n],\ \left(\mathsf{sk}_i^1, \mathsf{vk}_i^1\right) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

*We remark that this property implies that correctness and unforgeability holds for keys sampled by* $\mathsf{KeyGen}(1^\lambda, \mathsf{td})$ *for a random* $\mathsf{td} \leftarrow \{0,1\}^\lambda$.

**Extraction Correctness.** *For any* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{Extract}(\mathsf{td}, \mathsf{vk}) = 1 \quad : \quad \begin{array}{l} \mathsf{td} \leftarrow \{0,1\}^\lambda \\ (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{td}) \end{array}\right] = 1.$$

*Additionally, for any poly-size adversary* $\mathcal{A}$*, and for any polynomial* $n(\lambda)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{l} \mathsf{Extract}(\mathsf{td}, \mathsf{vk}) = 1 \wedge \\ \mathsf{vk} \notin \left\{\mathsf{vk}_i\right\}_{i\in[n]} \end{array} \quad : \quad \begin{array}{l} \mathsf{td} \leftarrow \{0,1\}^\lambda \\ \forall i \in [n],\ (\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{td}) \\ \mathsf{vk} \leftarrow \mathcal{A}(1^\lambda, \left\{\mathsf{sk}_i^b, \mathsf{vk}_i^b\right\}_{i\in[n]}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We note that given a signature scheme $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, we can construct a signature scheme $\mathsf{S}'$ with trapdoor keys, by appending a random padding $\mathsf{pad} \leftarrow \{0,1\}^\lambda$ to its verification keys. To generate a trapdoor key $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{KeyGen}'(1^\lambda, \mathsf{td})$, we instead append a pseudorandom padding $\mathsf{pad} = \mathsf{PRF}(\mathsf{td}, \mathsf{vk})$. Extraction is implemented by comparing pad to $\mathsf{PRF}(\mathsf{td}, \mathsf{vk})$, and the required properties follow from PRF security. We state the following lemma without proof for the above stated construction, and note that the proof follows in a straightforward manner akin to the construction of message authentication codes (MAC) from PRF: (i) extraction correctness follows from the guarantee that the PRF behaves like a MAC on the verification key; and (ii) key indistinguishability follows from the pseudorandomness of the PRF.

**Lemma 5.7.** *Assuming the existence of a* PRF *and a signature scheme, there exists a signature scheme with trapdoor keys.*

### 5.4.2 Construction

We now describe our construction that uses BARGs with functional subset extraction and a signature scheme with trapdoor keys. Specifically, let $F = \{F_\lambda\}$ be a family of monotone policies. We construct an $F$-aggregation scheme with weak unforgeability for S (Definition 4.2) from the following building blocks:

- A signature scheme with trapdoor keys $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ (Definition Definition 5.6).
- A $F$-batch argument with $G$-functional subset extraction for relation $\mathcal{R}_{\mathsf{full}}$ with hashed inputs (Definition 5.3 and Remark 5.1)

$$(\mathsf{Setup}_{\mathsf{BARG}}, \mathcal{P}_{\mathsf{BARG}}, \mathcal{V}_{\mathsf{BARG}}, \mathsf{Extract}_{\mathsf{BARG}}) ,$$

  where $G = \{G_\lambda\}_{\lambda \in \mathbb{N}}$ is a family of polynomial size circuits, with a polynomial bound set to support computing the Extract algorithm of the underlying signature scheme with trapdoor keys.
- A hash family with local opening (Definition 3.2)

$$(\mathsf{Gen}_{\mathsf{HT}}, \mathsf{Hash}_{\mathsf{HT}}, \mathsf{Open}_{\mathsf{HT}}, \mathsf{Verify}_{\mathsf{HT}}) .$$

We describe the aggregate signature algorithms. The construction is identical to Section 5.3, except that the BARG algorithms now receive a function and tag as needed, where we program the null function (later modified in the security proof), and our tag is the tuple of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$ (or the hashed aggregate verification keys $\widehat{\mathsf{vk}}$ for the verifier, following the syntax in Remark 5.1). This is given in full detail below, and we mark the changes from the previous transformation via a highlight. We prove the following theorem.

**Theorem 5.8.** *Assuming the existence of a digital signature scheme with trapdoor keys, hash family and $F$-batch argument with $G$-functional subset extraction, the construction given in Fig. 2 is an $F$-aggregation scheme for* $\mathsf{S}$.

**Remark 5.3.** The proof of Theorem 5.8 can be extended to extended to give a stronger notion of aggregated signatures where each party can sign a different message (see Remark 4.1). Moreover, if the the verification algorithm is given as input a succinct representation of the messages signed, its running time grows with the representation size instead of $k$. In more detail, we can support a representation of the messages $\{\mathsf{m}_i\}_{i \in [k]}$, either as a Turing machine that given input $i \in [k]$ outputs $\mathsf{m}_i$ in time $\mathsf{poly}(\lambda)$, or as the root of hash tree over the messages. See Remark 5.2 for more details.

**Proof of Theorem 5.8.**

**Correctness.** Follows directly from the completeness and correctness properties of the underlying batch argument, digital signature scheme and hash family with local opening.

**Efficiency.** Follows directly from the efficiency of the underlying primitives, and since each batch argument witness is of length $\mathsf{poly}(\lambda)$.

**Weakly Unforgeable.** Fix any poly-size adversary $\mathcal{A}$. Assume towards contradiction that there exists a polynomial poly such that for infinitely many $\lambda \in \mathbb{N}$, there exists $f \in F_\lambda$ such that $\mathcal{A}$ wins the weak unforgeability game with probability $> \frac{1}{\mathsf{poly}(\lambda)}$.

We consider an alternative setup procedure $\mathsf{Setup}(1^\lambda, \mathsf{td}_{\mathsf{S}})$, and an alternative challenger $C'$ for the weak unforgeability game.

$\mathsf{Setup}(1^\lambda, \mathsf{td}_{\mathsf{S}})$ algorithm does the following:

$\mathsf{Setup}(1^\lambda)$:

1. Generate $\mathsf{hk} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$.
2. Let $g$ be a null function (we can use any arbitrary function).
3. Generate $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Setup}_{\mathsf{BARG}}(1^\lambda, g)$.
4. Output $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.

$\mathsf{KeyAgg}(\mathsf{crs}, \{\mathsf{vk}_i\}_{i\in[k]})$:

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.
2. Output $\widehat{\mathsf{vk}} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$.

$\mathsf{SigAgg}(\mathsf{crs}, f, \{\mathsf{vk}_i, \sigma_i\}_{i\in[k]}, \mathsf{m})$:

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.
2. Compute $\widehat{\mathsf{vk}} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$.
3. Let $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$, and define the Turing machine $M(z, j, w_j)$ which operates as follows:
   - Parse $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$.
   - Parse $w_j = (\rho_j, \mathsf{vk}_j, \sigma_j)$.
   - Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}, \widehat{\mathsf{vk}}, j, \mathsf{vk}_j, \rho_j) = 1$.
   - Check that $\mathsf{Verify}(\mathsf{vk}_j, \mathsf{m}, \sigma_j) = 1$.
4. For every $j \in [k]$, construct a witness $w_j$ for $x$, using the $\mathsf{Open}_{\mathsf{HT}}$ algorithm to produce the appropriate openings. Let $w = (w_1, \ldots, w_k)$.
5. Let $y = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$.
6. Output $\widehat{\sigma} = \mathcal{P}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, y, f, M, z, 1^T, w)$.

$\mathsf{AggVerify}(\mathsf{crs}_{\mathcal{V}}, f, \widehat{\mathsf{vk}}, \mathsf{m}, \widehat{\sigma})$:

1. Parse $\mathsf{crs}_{\mathcal{V}} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.
2. Define $M, T$ as above and $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$, and let $x = (f, M, z, T)$.
3. Output $\mathcal{V}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, \widehat{\mathsf{vk}}, x, \widehat{\sigma})$.

Figure 2: Weakly Unforgeable Aggregate Signatures for Monotone Circuits $f$ from BARGs with Functional Subset Extraction for $f$

1. Generate $\mathsf{hk} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$.
2. Let $g_{\mathsf{td}_{\mathsf{S}}}$ be a function that does the following given input $y$:
   (a) Parse $y = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$.
   (b) Output $J = \{i \in [k] \ : \ \mathsf{Extract}(\mathsf{td}_{\mathsf{S}}, \mathsf{vk}_i) = 1\}$.
3. Generate $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Setup}_{\mathsf{BARG}}(1^\lambda, g_{\mathsf{td}_{\mathsf{S}}})$.
4. Output $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$.

The alternative challenger $C'$ for the weak unforgeability game does the following:

1. Sample $\mathsf{td}_S \leftarrow \{0,1\}^\lambda$.
2. Sample $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{td}_S)$ and give crs to $\mathcal{A}$.
3. Answer the adversary's queries as follows:
   - Verification key queries: generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{td}_S)$ (using the trapdoor key generation algorithm rather than the honest key generation algorithm).
   - Signing queries are unchanged.

We claim that in the weak unforgeability game of $\mathcal{A}$ against the alternative challenger $C'$, the adversary still wins with probability $> \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)$. This follows from the key indistinguishability properties of the underlying batch argument and digital signature with trapdoor keys.

We now construct an adversary $\mathcal{B}$ against the underlying digital signature scheme, similarly to the one in the proof of Theorem 5.4, Given the security parameter $1^\lambda$, a verification key $\mathsf{vk}^*$, and oracle access to $\mathsf{Sign}(\mathsf{sk}^*, \cdot)$, $\mathcal{B}$ does the following:

1. Let $Q(\lambda)$ be a polynomial upper bound on the number of verification key queries that $\mathcal{A}$ makes. Sample $i^* \leftarrow [Q]$.
2. Simulate the aggregate signature unforgeability game against $\mathcal{A}$.
   - Sample $\mathsf{td}_S \leftarrow \{0,1\}^\lambda$.
   - Generate $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{td}_S)$ and additionally save $\mathsf{td}_{\mathsf{BARG}}$. Give crs to $\mathcal{A}$.
   - Whenever $\mathcal{A}$ makes a verification key query, if it is the $i^*$th query, give it $\mathsf{vk}^*$. Otherwise, sample $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{td}_S)$ and give it vk.
   - Whenever $\mathcal{A}$ makes a signing query, if it requests to sign under $\mathsf{vk}^*$, query the signing oracle $\mathsf{Sign}(\mathsf{sk}^*, \cdot)$ and give the result. Otherwise, sign with the signing key sk that was sampled together with vk.
3. At the end of the game, let $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \mathsf{m}, \widehat{\sigma}$ be $\mathcal{A}$'s output.
4. Extract $(i, w_i) \leftarrow \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}_{\mathsf{BARG}}, y, \widehat{\sigma})$ where $y = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$.
5. Parse $w_i = (\rho_i, \mathsf{vk}'_i, \sigma_i)$ and output $\mathsf{m}, \sigma_i$.

Thus, as in the proof of Theorem 5.4 and since unforgeability holds for trapdoor keys, it suffices to prove the following claim:

**Claim 5.9.** *In the experiment of running $\mathcal{B}$, considering the extracted $i, w_i = (\rho_i, \mathsf{vk}'_i, \sigma_i)$, let* GOOD *be the event that $\mathsf{vk}'_i$ is a key that $\mathcal{A}$ received via a verification key query, and $\mathcal{A}$ did not make a signing query for $\mathsf{vk}'_i$ and m, and $\mathsf{Verify}(\mathsf{vk}'_i, \mathsf{m}, \sigma_i) = 1$. Then, we have $\Pr[\mathrm{GOOD}] > \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)$.*

**Proof of Claim 5.9.** The claim follows from the functional subset extraction of the underlying batch argument, together with the extraction correctness of the digital signature with trapdoor keys.

For a given trapdoor $\mathsf{td}_S \in \{0,1\}^\lambda$ we consider a cheating prover $\mathcal{P}^*_{\mathsf{td}_S}$, that does the following given $\mathsf{crs}_{\mathsf{BARG}}$:

1. Simulate the aggregate signature unforgeability game against $\mathcal{A}$ with challenger $C'$, giving it the common reference string $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{BARG}})$ for some sampled $\mathsf{hk} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$ and using $\mathsf{KeyGen}(1^\lambda, \mathsf{td}_{\mathsf{S}})$ and Sign as required.
2. At the end of the game, let $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k), \mathsf{m}, \widehat{\sigma}$ be $\mathcal{A}$'s output.
3. Let $M$ as in SigAgg and $z = (\mathsf{hk}, \widehat{\mathsf{vk}}, \mathsf{m})$ where $\widehat{\mathsf{vk}} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$.
4. Let $y = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$.
5. Output $M, z, \widehat{\sigma}, y$.

Let EXP be the experiment defined in the functional subset extraction requirement:

- Generate $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Setup}_{\mathsf{BARG}}(1^\lambda, g_{\mathsf{td}_{\mathsf{S}}})$ (where $g_{\mathsf{td}_{\mathsf{S}}}$ is defined as in $\mathsf{Setup}(1^\lambda, \mathsf{td}_{\mathsf{S}})$.
- Run the cheating prover above and obtain $(M, z, \pi, y) \leftarrow \mathcal{P}^*_{\mathsf{td}}(\mathsf{crs}_{\mathsf{BARG}})$.
- Let $x = (f_\lambda, M, z, T)$ and $J = g_{\mathsf{td}_{\mathsf{S}}}(y)$.
- Extract $(i, w_i) \leftarrow \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}_{\mathsf{BARG}}, y, \pi)$.

In the above experiment, consider the event that $\mathcal{A}$ wins the weak unforgeability game against the alternative challenger $C'$. In this event, by definition we have that (i) $f(b_1, \ldots, b_k) = 0$ where $b_i = 0$ if $\mathsf{vk}_i$ is a key that $\mathcal{A}$ received via a verification key query, (ii) $\mathcal{V}_{\mathsf{BARG}}(\mathsf{crs}_{\mathsf{BARG}}, y, x, \pi) = 1$, and (iii) $\mathcal{A}$ did not make any signing query for $\mathsf{m}$.

By extraction correctness of the digital signature with trapdoor keys, we have that except with negligible probability, $\mathsf{Extract}(\mathsf{td}_{\mathsf{S}}, \mathsf{vk}_i) = b_i$ for all $i$. Indeed, if $\mathsf{vk}_i$ was obtained via a verification key query then it was generated with $\mathsf{KeyGen}(1^\lambda, \mathsf{td}_{\mathsf{S}})$ and so $\mathsf{Extract}(\mathsf{td}_{\mathsf{S}}, \mathsf{vk}_i) = 1$ with probability 1, and otherwise we have that $\mathsf{Extract}(\mathsf{td}_{\mathsf{S}}, \mathsf{vk}_i) = 1$ with negligible probability.

Therefore, conditioned on $\mathcal{A}$ winning, we have that except with negligible probability, $f(\mathbb{1}_{[k]\setminus J}) = f(b_1, \ldots, b_k) = 0$. Since we know that $\mathcal{A}$ wins the game against $C$ with probability $> \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)$, we get

$$\Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}_{\mathsf{BARG}}(\mathsf{crs}_{\mathcal{V}}, y, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] > \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)\,.$$

We now use the functional subset extraction of the underlying batch argument. We obtain

$$\Pr_{\mathsf{EXP}}\left[\; i \in J \;\wedge\; M(z, i, w_i) = 1 \;\right] > \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)\,,$$

and this event implies GOOD (as in Theorem 5.4), which concludes the proof of Claim 5.9 and Theorem 5.8.

# 6 Composable Verifiable Private Information Retrieval for Policies

In this section, we introduce and construct the main building block in the construction of our BARGs with adaptive subset extraction for read-once monotone policies, composable verifiable private information retrieval (vPIR). Our notion of composable vPIR extends the notion of vPIR to handle a composable definition of security, and additionally monotone policies, both of which is necessary to construct our BARGs with adaptive subset extraction for read-once monotone policies in Section 7.

We start by defining of composable vPIR for policies in Section 6.1. In Section 6.2 we give a construction of vPIR that supports our extensions, and in Section 6.3 we extend the analysis of [BDKP22] to show that this construction satisfies our new composable security definition. We first briefly describe vPIR for policies, and how it differs from the notion considered in [BDKP22].

vPIR **for policies.** The work of [BDKP22] considers vPIR for bounded space global database constraints. Such constraints are evaluated by reading the rows of the database one by one and maintaining a state of bounded size between rows. In the adaptive setting, where the adversary may choose the constraint as a function of the vPIR query, the security of their construction degrades exponentially with the description length of the constraint (given by a program that updates state). However, in the non-adaptive setting, their construction does not incur such loss.

We consider a particular type of bounded space global constraints that are suitable for describing policies. The description of such constraints can be separated into two parts: a program $\Gamma$ and a vector of instances $X$, one for every row in the database. The program updates the state as a function of both the current row and the corresponding instance. Looking ahead, in our application the instances may be chosen adaptively. Therefore, relying on the construction from [BDKP22] would result in exponential security loss. However, in our instantiation the security loss is only exponential in the description length of the program $\Gamma$ and not of the instances. Moreover, in the adaptive setting, the simulator in the definition from [BDKP22] is required to produce the entire constraint $(\Gamma, X)$ (this constraint together with the simulated client's output row $D[t]$ is required to be indistinguishable from the constraint and output in the real world), while our notion is relaxed: we only simulate the constraint $\Gamma$ and the relevant instance $X[t]$.

## 6.1 Definition

We define a simulation security notion for multiple parallel executions of vPIR. While one could consider general forms of composition, we focus on the setting where a single client is sending a single vPIR query to $p$ (potentially colluding) servers, each holding a different database and constraint. In this setting, we require the simulator to produce the output of the client (including the database row and constraint) in all $p$ interactions.

For simplicity, in Definition 6.1 we simplify this further and only consider the case of $p = 2$ colluding servers, which suffices for our applications to BARGs with adaptive subset extraction and aggregate signatures. However we note that the definition can be naturally extended to define $p$-composable vPIR.

**Read-once bounded-space policy constraints.** For parameters $T, S$, we define a read-once bounded-space constraint represented as a Turing machine $\Gamma \in \{0,1\}^N$ and instances $X = (x_i)_{i\in[k]} \in \{0,1\}^{k\times n}$. We say that a database $D = (r_i)_{i\in[k]} \in \{0,1\}^{k\times w}$ satisfies the policy defined by $(\Gamma, X)$ if and only if for every $i \in [k-1]$ there exists $c_i \in \{0,1\}^S$ such that $\Gamma(c_{i-1}, x_i, r_i)$ outputs $c_i$ within $T$ steps where $c_0, c_k$ are some fixed starting and accepting configurations. We denote by $U_{T,S}(\Gamma, X, D)$ the bit that indicates whether or not $D$ satisfies the policy $(\Gamma, X)$.

**Syntax.** A verifiable private information retrieval (vPIR) scheme for policies consists of the following polynomial-time algorithms:

$\mathsf{Query}(1^\lambda, t) \to (\mathsf{dk}, \mathsf{vk}, q)$**.** This is a probabilistic algorithm that takes as input the security parameter $1^\lambda$, and a row index $t \in [2^\lambda]$. It outputs a decryption key $\mathsf{dk}$, a verification key $\mathsf{vk}$ and a query $q$.

$\mathsf{Answer}(D, 1^T, \Gamma, X, q) \to a$. This is a deterministic algorithm that takes as input a database $D \in \{0,1\}^{k \times \omega}$, a time-bound $1^T$, a constraint $\Gamma$, instances $X \in \{0,1\}^{k \times n}$, and a query $q$. It outputs an answer $a$.

$\mathsf{Dec}(\mathsf{dk}, a) \to r$. This is a deterministic algorithm that takes as input a decryption key $\mathsf{dk}$, and an answer $a$. It outputs a row $r \in \{0,1\}^{\omega}$.

$\mathsf{Verify}(\mathsf{vk}, \Gamma, X, a) \to 0/1$. This is a deterministic algorithm that takes as input a verification key $\mathsf{vk}$, a constraint $\Gamma$, instances $X \in \{0,1\}^{k \times n}$, and an answer $a$. It outputs a bit (1 to accept, 0 to reject).

**Remark 6.1** (Index instances)**.** In our constructions in Section 7, we only need to consider instances $X$ of the form $X = (x, i)_{i \in [k]}$ for some $x \in \{0,1\}^n$ that is the same in all rows. We note that in this case, the Answer, Verify algorithms may receive just $x$ rather than $X = (x, i)_{i \in [k]}$ (and so the verification time is independent of $k$).

**Definition 6.1.** *A $\Lambda$-secure 2-composable* vPIR *scheme for policies* (Query, Answer, Dec, Verify) *is required to satisfy the following properties:*

**Completeness.** *For any $\lambda \in \mathbb{N}$, any $T, N, S, k, n, \omega \leq 2^\lambda$, database $D \in \{0,1\}^{k \times \omega}$, row index $t \in [k]$, constraint $\Gamma \in \{0,1\}^N$ and instances $X \in \{0,1\}^{k \times n}$ such that $U_{T,S}(\Gamma, X, D) = 1$,*

$$\Pr\left[\begin{array}{ll} \mathsf{Dec}(\mathsf{dk}, a) = D[t] \wedge & (\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t) \\ \mathsf{Verify}(\mathsf{vk}, \Gamma, X, a) = 1 \quad : & a \leftarrow \mathsf{Answer}(D, 1^T, \Gamma, X, q) \end{array}\right] = 1 \,.$$

**Efficiency.** *In the completeness experiment above, $|\mathsf{vk}|+|q|+|a| \leq \omega \cdot \mathsf{poly}(\lambda)$.*

**$\Lambda$-Privacy.** *For any $\mathsf{poly}(\Lambda)$-size adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}$ such that for any $\lambda \in \mathbb{N}$ and row indices $t_0, t_1 \in [2^\lambda]$,*

$$\Pr\left[\begin{array}{ll} \mathcal{A}(\mathsf{vk}, q) = b \quad : & b \leftarrow \{0,1\} \\ & (\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\Lambda) \,.$$

**2-Composable $\Lambda$-simulation security.** *For any functions $T(\lambda), N(\lambda), n(\lambda), k(\lambda), \omega(\lambda) \leq \Lambda(\lambda)$, function $S(\lambda) = O(\log \Lambda)$, any $\mathsf{poly}(\Lambda)$-size adversary $\mathcal{A}$ and polynomial $P$ there exists a $\mathsf{poly}(\Lambda)$-size simulator $\mathsf{Sim}$ such that for any $\mathsf{poly}(\Lambda)$-size distinguisher $\mathcal{D}$, $\lambda \in \mathbb{N}$, constraints $\Gamma, \Gamma' \in \{0,1\}^N$, and row index $t \in [k]$,*

$$\left|\Pr\left[\ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma, \Gamma', t)) = 1\ \right] - \Pr\left[\ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma, \Gamma', t)) = 1\ \right]\right| < \frac{1}{P(\Lambda)} \,,$$

*where the experiments $\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma, \Gamma', t)$ and $\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma, \Gamma', t)$ are defined as follows:*

$\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma, \Gamma', t)$**:**

- *Generate a query* $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t)$.
- *Run the adversary and obtain* $(X, X', a, a') \leftarrow \mathcal{A}(\mathsf{vk}, q)$.
- *If* $\mathsf{Verify}(\mathsf{vk}, \Gamma, X, a) = 1$ *and* $\mathsf{Verify}(\mathsf{vk}, \Gamma', X', a') = 1$ *output* $(X[t], X'[t], \mathsf{Dec}(\mathsf{dk}, a), \mathsf{Dec}(\mathsf{dk}, a'))$.
- *Otherwise output* $\perp$.

$\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma, \Gamma', t)$**:**

- *Run the simulator and obtain* $(X, X', D, D') \leftarrow \mathsf{Sim}(\lambda, \Gamma, \Gamma')$.
- *If* $U_{T,S}(\Gamma, X, D) = 1$ *and* $U_{T,S}(\Gamma', X', D') = 1$ *output* $(X[t], X'[t], D[t], D'[t])$.
- *Otherwise output* $\perp$.

## 6.2 Construction

In this section we construct a 2-composable vPIR for policies. The construction is similar to the construction given in [BDKP22], slightly modified to handle our extensions. It uses the following building blocks:

- A $\Lambda$-secure hash family with local opening (Definition 3.2)

$$(\mathsf{Gen}_{\mathsf{HT}}, \mathsf{Hash}_{\mathsf{HT}}, \mathsf{Open}_{\mathsf{HT}}, \mathsf{Verify}_{\mathsf{HT}}).$$

- A $\Lambda$-secure seBARG scheme (Definition 3.5)

$$(\mathsf{Gen}_{\mathsf{seBARG}}, \mathcal{P}_{\mathsf{seBARG}}, \mathcal{V}_{\mathsf{seBARG}}, \mathsf{Extract}_{\mathsf{seBARG}}).$$

We are now ready to describe our 2-composable vPIR scheme for policies.

## 6.3 Analysis

In this section we analyze the construction given in Section 6.2, and prove the following theorem:

**Theorem 6.2.** *Assuming a $\Lambda$-secure hash family with local opening and a $\Lambda$-secure somewhere-extractable batch argument scheme, there exists a $\Lambda$-secure 2-composable* vPIR *scheme for policies.*

Our analysis, detailed below, is similar to the analysis of simulation secure vPIR in [BDKP22], with some parts taken verbatim from the full version. We extend it to handle policy constraints, and 2-composable simulation security.

Before proceeding with the formal proof, we describe the high-level ideas in our analysis. We first recall the simulator given in [BDKP22]. It simulates an accepting database using coupling: for each row index in the database $i \in [k]$, it repeatedly queries the server for this index and constructs a list $L_i$ of tuples $(c_{i-1}, r_i)$, where $c_{i-1}$ is the intermediate configuration of the bounded-space constraint $\Gamma$ before reading the corresponding row $r_i$. Then, it couples the lists $L_1, \ldots, L_k$ to create a list $L$ of full databases, so that the intermediate configurations "connect" to a consistent execution of $\Gamma$. Finally it samples a random database from this list.

We now describe the changes in our analysis. To support policies (where we also need to simulate instances $X \in \{0,1\}^{k \times n}$), our simulator constructs the list $L_i$ so that it contains tuples $(c_{i-1}, x_i, r_i)$ (where $x_i$ is the $i$th instance given by the prover in this query), then couples the lists in the same way. To extend the analysis to composition of $p$ parallel executions, each element in the list $L_i$ is a $p$-tuple of $(c^j_{i-1}, x^j_i, r^j_i)_{j \in [p]}$, with one configuration-instance-row tuple corresponding to each server. We then couple $L_1, \ldots, L_k$ to obtain a list $L$ such that each individual database connects.

We note that, in order for the coupling argument to work, we need to assume hardness assumptions that are exponential in the support size of the function according to which we're coupling (which, in our case, is the combined size of all intermediate configurations $cf_i$ in the $p$ parallel executions, for any given $i$). In [BDKP22] this is what limits the space $S$ of the constraint to $O(\log \Lambda)$, and in our case the limit is $p \cdot S = O(\log \Lambda)$, assuming $\Lambda$-hardness of the underlying building blocks.

**Proof of Theorem 6.2.**

**Completeness.** Follows directly from the completeness of the underlying seBARG and hash family with local opening.

$\mathsf{Query}(1^\lambda, t)$:

1. Generate $\mathsf{hk}_{\mathsf{HT}} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$.
2. Generate $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, I)$, where $I = \{1, t\}$.
3. Output $\mathsf{dk} = \mathsf{td}_{\mathsf{seBARG}}, \mathsf{vk} = q = (\mathsf{hk}_{\mathsf{HT}}, \mathsf{crs}_{\mathsf{seBARG}})$.

$\mathsf{Answer}(D, 1^T, \Gamma, X, q)$:

1. Parse $q = (\mathsf{hk}_{\mathsf{HT}}, \mathsf{crs}_{\mathsf{seBARG}})$.
2. Parse $D = (r_i)_{i\in[k]} \in \{0,1\}^{k\times\omega}$ and $X = (x_i)_{i\in[k]} \in \{0,1\}^{k\times n}$.
3. Let $c_0$ be the starting configuration, and for every $1 \le i \le k$ compute $c_i = \Gamma(c_{i-1}, x_i, r_i)$.
4. Compute $\mathsf{rt} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, (c_0, \ldots, c_k))$.
5. Define an instance $Y = (M, z, k, T')$ of BatchIndexTMSAT. The input $z$ is defined as $z = (X, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$. The batch size is set to $k$. The Turing machine $M(z, j, w_j)$ is defined to operate as follows:
   (a) Parse $z = (X, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$, and $X = (x_i)_{i\in[k]}$.
   (b) Parse $w_j = (r_j, c_{j-1}, c_j, \rho_{j-1}, \rho_j)$, where $m' = |w_j|$.
   (c) If $j = 1$, check that $c_0$ is the starting configuration. If $j = k$, check that $c_k$ is the accepting configuration.
   (d) Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j-1, c_{j-1}, \rho_{j-1}) = 1$ and $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j, c_j, \rho_j) = 1$.
   (e) Check that $\Gamma(c_{j-1}, x_j, r_j) = c_j$.

   The description length of $M$ is a constant. Finally, the time bound $T' = \mathsf{poly}(n, k, w, T, N, S)$ is set so that the pseudocode above terminates.
6. For every $j \in [k]$, construct a witness $(j, w_j)$ for $Y$, using the database $D = (r_i)_{i\in[k]}$, the computed configurations $c_0, \ldots, c_k$, and the $\mathsf{Open}_{\mathsf{HT}}$ algorithm to produce the appropriate openings.
7. Compute $\pi_{\mathsf{seBARG}} = \mathcal{P}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, M, z, 1^{T'}, w_1, \ldots, w_k)$.
8. Output $a = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.

$\mathsf{Dec}(\mathsf{dk}, a)$:

1. Parse $\mathsf{dk} = \mathsf{td}_{\mathsf{seBARG}}$ and $a = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
2. Extract $w_1, w_i = \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$.
3. Parse $w_i = (r_i, c_{i-1}, c_i, \rho_{i-1}, \rho_i)$.
4. Output $r_i$.

$\mathsf{Verify}(\mathsf{vk}, \Gamma, X, a)$:

1. Parse $\mathsf{vk} = (\mathsf{hk}_{\mathsf{HT}}, \mathsf{crs}_{\mathsf{seBARG}})$ and $a = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
2. Define $Y = (M, z, k, T')$ as in Answer.
3. Output $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, Y, \pi_{\mathsf{seBARG}})$.

Figure 3: Construction of 2-Composable vPIR Scheme for Policies

**Efficiency.** We have $|\mathsf{vk}|+|q|+|a|= |\mathsf{hk}_{\mathsf{HT}}|+|\mathsf{rt}|+|\mathsf{crs}_{\mathsf{seBARG}}|+|\pi_{\mathsf{seBARG}}|$.

- By succinctness of the hash family with local opening, $|\mathsf{hk}_{\mathsf{HT}}|+|\mathsf{rt}|= \mathsf{poly}(\lambda)$.
- By the seBARG efficiency,

$$|\mathsf{crs}_{\mathsf{seBARG}}|+|\pi_{\mathsf{seBARG}}|\leq (\omega + 2S + 2|\rho|) \cdot \mathsf{poly}(\lambda) = \omega \cdot \mathsf{poly}(\lambda) \ .$$

So indeed $|\mathsf{vk}|+|q|+|a|\leq \omega \cdot \mathsf{poly}(\lambda)$.

**$\Lambda$-Privacy.** Follows directly from the $\Lambda$-index hiding property of the underlying seBARG.

**$\Lambda$-Composable simulation security.** Fix functions $T(\lambda), N(\lambda), n(\lambda)k(\lambda), \omega(\lambda) \leq \Lambda(\lambda)$ and $S = O(\log \Lambda)$, poly-size adversary $\mathcal{A}$, and polynomial $P$.

We recall the following lemma from [BDKP22]:

**Lemma 6.3** (Coupling [BDKP22])**.** *For $\ell \in \mathbb{N}$ and sets $X_1, X_2$, let $\left\{L_i \in X_i^\ell\right\}_{i\in[2]}$ be a pair of lists and let $\{f_i : X_i \to \{0,1\}^*\}_{i\in[2]}$ be a pair of polynomial-time computable functions. Then there exists a list of pairs $L' = (X_1 \times X_2)^\ell$ (the coupling of $L_1$ and $L_2$ with respect to $f_1, f_2$) such that for $x_1 \leftarrow L_1, x_2 \leftarrow L_2$ and $(x'_1, x'_2) \leftarrow L'$ it holds that $x_i \equiv x'_i$ and:*

$$\Pr\left[\ f_1(x'_1) = f_2(x'_2)\ \right] = 1 - \mathrm{SD}\left(f_1(x_1), f_2(x_2)\right) \ .$$

*Moreover, $L'$ can be computed from $L_1, L_2$ in polynomial time.*

We are now ready to describe the simulator $\mathsf{Sim}$. Given the security parameter $\lambda$ and constraints $\Gamma^0, \Gamma^1$, it proceeds as follows:

1. Set $\ell = (20kP(\Lambda))^2 \cdot 2^{4S} \cdot \Lambda(\lambda)$ and $\epsilon = \frac{1}{2P(\Lambda(\lambda))}$ .
2. Generate $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t = 1)$.
3. Run the adversary and obtain $(X^0, X^1, a^0, a^1) \leftarrow \mathcal{A}(\mathsf{vk}, q)$.
4. If $\mathsf{Verify}(\mathsf{vk}, \Gamma^0, X^0, a^0) = 0$ or $\mathsf{Verify}(\mathsf{vk}, \Gamma^1, X^1, a^1) = 0$, output $\perp$.
5. For every $i \in [k]$:
   (a) Set $L_i$ to be an empty list.
   (b) Repeat the following $\Lambda(\lambda) \cdot \ell/\epsilon$ times:
      i. Generate $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, i)$. Parse $\mathsf{dk} = \mathsf{td}_{\mathsf{seBARG}}$.
      ii. Run the adversary and obtain $(X^0, X^1, a^0, a^1) \leftarrow \mathcal{A}(\mathsf{vk}, q)$.
      iii. If $\mathsf{Verify}(\mathsf{vk}, \Gamma^0, X^0, a^0) = 0$ or $\mathsf{Verify}(\mathsf{vk}, \Gamma^1, X^1, a^1) = 0$, return to Item 5b.
      iv. For $b \in \{0,1\}$, parse $X^b = (x_i^b)_{i\in[k]}$ and $a^b = (\mathsf{rt}^b, \pi^b_{\mathsf{seBARG}})$.
      v. For $b \in \{0,1\}$, extract $w_1^b, w_i^b \leftarrow \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi^b_{\mathsf{seBARG}})$.
      vi. For $b \in \{0,1\}$, parse $w_i^b = (r_i^b, c_{i-1}^b, c_i^b, \rho_{i-1}^b, \rho_i^b)$.
      vii. Append $(c_{i-1}^b, x_i^b, r_i^b)_{b\in\{0,1\}}$ to $L_i$.
   (c) If $|L_i| < \ell$ output $\perp$. Otherwise, truncate its length to exactly $\ell$.

6. Set $L'_1 = L_1 \in \{0,1\}^{m' \cdot \ell}$, where $m'$ is the size of the witness used in the BARG.
7. For every $i \in [k-1]$:
   (a) Let $f'_i \colon \{0,1\}^{m' \cdot i} \to \{0,1\}^S$ and $f_{i+1} \colon \{0,1\}^{m'} \to \{0,1\}^S$ be the following functions:
   $$f'_i((c^b_{j-1}, x^b_j, r^b_j)_{j \in [i], b \in \{0,1\}}) = (\Gamma(c^0_{i-1}, x^0_i, r^0_i), \Gamma(c^1_{i-1}, x^1_i, r^1_i)) \,,$$
   $$f_{i+1}((c^b_i, x^b_{i+1}, r^b_{i+1})_{b \in \{0,1\}}) = (c^0_i, c^1_i) \,.$$
   (b) Compute $L'_{i+1} \in \{0,1\}^{(m' \cdot (i+1)) \cdot \ell}$, the coupling of $L'_i$ and $L_{i+1}$ with respect to $f'_i, f_{i+1}$ given by Lemma 6.3.
8. Sample $(c^b_{i-1}, x^b_i, r^b_i)_{i \in [k], b \in \{0,1\}} \leftarrow S'_k$ (uniformly from the list).
9. For $b \in \{0,1\}$, set $X^b = (x^b_i)_{i \in [k]}$ and $D^b = (r^b_i)_{i \in [k]}$.
10. Output $(X^0, X^1, D^0, D^1)$.

Fix a poly($\Lambda$)-size distinguisher $\mathcal{D}$. Let $\lambda \in \mathbb{N}$, constraints $\Gamma^0, \Gamma^1$ and query $t \in [k]$, and denote the real and ideal experiments defined in Definition 6.1 by $\mathsf{EXP}_{\mathsf{Real}}, \mathsf{EXP}_{\mathsf{Ideal}}$. Additionally, we omit $\Gamma^0, \Gamma^1$ from the inputs of $\mathsf{Real}_{\mathcal{A}}, \mathsf{Ideal}_{\mathsf{Sim}}, \mathsf{Sim}$.

**Step 1: Bound the probability that Sim fails in Item 4.** In the real experiment, let $E_\perp$ be the event that $\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma^0, \Gamma^1, t) = \perp$. In the ideal experiment, let $E_\perp$ be the event that Sim fails and outputs $\perp$ in Item 4. We have

$$\left| \Pr_{\mathsf{EXP}_{\mathsf{Real}}} \left[ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda, t)) = 1 \right] - \Pr_{\mathsf{EXP}_{\mathsf{Ideal}}} \left[ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, t)) = 1 \right] \right| \tag{1}$$

$$\leq \left| \Pr_{\mathsf{EXP}_{\mathsf{Real}}} \left[ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda, t)) = 1 \wedge E_\perp \right] - \Pr_{\mathsf{EXP}_{\mathsf{Ideal}}} \left[ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, t)) = 1 \wedge E_\perp \right] \right| \tag{2}$$

$$+ \left| \Pr_{\mathsf{EXP}_{\mathsf{Real}}} \left[ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda, t)) = 1 \wedge \neg E_\perp \right] - \Pr_{\mathsf{EXP}_{\mathsf{Ideal}}} \left[ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, t)) = 1 \wedge \neg E_\perp \right] \right| . \tag{3}$$

To bound Eq. (2), we observe that $E_\perp$ in the real and ideal experiments are the events that $\mathcal{A}$ fails when queried on $t$, and when queried on 1. Therefore, by $\Lambda$-privacy, we have

$$\left| \Pr_{\mathsf{EXP}_{\mathsf{Real}}} \left[ E_\perp \right] - \Pr_{\mathsf{EXP}_{\mathsf{Ideal}}} \left[ E_\perp \right] \right| \leq \mathsf{negl}(\Lambda) \,,$$

which gives us the same bound for Eq. (2) (since in the case of $E_\perp$, both $\mathsf{Real}_{\mathcal{P}^*}(\lambda, t) = \perp$ and $\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, t) = \perp$, so $\mathcal{D}$ has the same output).

To show that Eq. (1) is at most $\frac{1}{P(\Lambda)}$, it suffices to show that Eq. (3) is at most $\frac{1}{2P(\Lambda)}$. First, note that if $\Pr_{\mathsf{EXP}_{\mathsf{Real}}} \left[ \neg E_\perp \right] \leq \frac{1}{5P(\Lambda)}$, then we immediately get the required bound, since by $\Lambda$-privacy we also have $\Pr_{\mathsf{EXP}_{\mathsf{Ideal}}} \left[ \neg E_\perp \right] \leq \frac{1}{5P(\Lambda)} + \mathsf{negl}(\Lambda)$.

Therefore, in the rest of the proof we assume wlog that $\Pr_{\mathsf{EXP}_{\mathsf{Real}}} \left[ \neg E_\perp \right] > \frac{1}{5P(\Lambda)}$, and our goal is to bound Eq. (3).

**Step 2: Bound the probability that** Sim **fails in Item 5c.** Let SHORT be the event that in the $\mathsf{EXP}_{\mathsf{Ideal}}$ experiment, one of the lists $L_i$ is shorter than $\ell$ and Sim outputs $\perp$ in Item 5c. We show that

$$\Pr_{\mathsf{EXP}_{\mathsf{Ideal}}}\left[\ \mathsf{SHORT} \ : \ \neg E_{\perp}\ \right] \leq \mathsf{negl}(\Lambda)\,.$$

Indeed, since we assumed (wlog, as shown in step 1) $\Pr_{\mathsf{EXP}_{\mathsf{Real}}}\left[\ \neg E_{\perp}\ \right] > \frac{1}{5P(\Lambda)}$, by seBARG index hiding we also have that the probability of appending a new item to the list in a single iteration of Item 5b is $> \frac{1}{5P(\Lambda)} - \mathsf{negl}(\Lambda)$. In $\Lambda(\lambda)\cdot 1/\epsilon$ iterations, the probability that we fail to append at least one item is $\leq 2^{-O(\Lambda)}$.

Thus, in the full $\Lambda(\lambda)\cdot \ell/\epsilon$ iterations for each $L_i$, by a union bound, the resulting $L_i$ would be shorter than $\ell$ with probability $\leq \ell \cdot 2^{-O(\Lambda)}$, and the probability that any $L_i$ is too short is $\leq k\cdot \ell \cdot 2^{-O(\Lambda)}$ which is negligible, as desired.

**Step 3: Bound the probability that** Sim**'s output is invalid.** Let BAD be the event that in the $\mathsf{EXP}_{\mathsf{Ideal}}$ experiment, Sim's output was not $\perp$ but $\mathsf{Ideal}(\lambda, t) = \perp$. This occurs when Sim outputs $X^0, X^1, D^0, D^1$ (which are not $\perp$) such that $U_\lambda(\Gamma^0, X^0, D^0) = 0$ or $U_\lambda(\Gamma^1, X^1, D^1) = 0$.

We show that conditioned on $\mathsf{Sim}(1^\lambda) \neq \perp$ in the $\mathsf{EXP}_{\mathsf{Ideal}}$ experiment (in other words, conditioned on $\neg E_\perp \wedge \neg\mathsf{SHORT}$), BAD occurs with probability $\leq \frac{1}{6P(\Lambda)}$. That is,

$$\Pr_{\mathsf{EXP}_{\mathsf{Ideal}}}\left[\begin{array}{l}\forall b\in\{0,1\},\ (c_0^b, c_k^b) = (\tilde{c}_0, \tilde{c}_k)\ \wedge \\ \forall b\in\{0,1\}, i\in[k],\ \Gamma(c_{i-1}^b, x_i^b, r_i^b) = c_i^b\end{array} \ : \ \mathsf{Sim}(1^\lambda)\neq\perp\ \right] \geq 1 - \frac{1}{6P(\Lambda)}\,,$$

where $\tilde{c}_0, \tilde{c}_k$ are the fixed correct starting and accepting configurations.

We recall that $(c_{i-1}^b, x_i^b, r_i^b)_{i\in[k], b\in\{0,1\}}$ are sampled by Sim from the coupled list $S'_k$, in which the marginal distribution of each tuple is identical to the distribution of $L_i$. Additionally we denote $c_k^b = \Gamma(c_{k-1}^b, x_k^b, r_k^b)$.

**Step 3a: starting and ending configurations.** We claim that for all $b\in\{0,1\}$, $(c_0^b, c_k^b) = (\tilde{c}_0, \tilde{c}_k)$ occurs with probability $1-\mathsf{negl}(\Lambda)$, from the underlying seBARG somewhere argument of knowledge. Indeed, when appending items to $L_i$ we check that $\mathsf{Verify}(\mathsf{vk}, \Gamma^b, X^b, a^b) = 0$, which implies that except with negligible probability the extracted witness $w_i$ satisfies $M(z, i, w_i) = 1$. For $i = 1$ and $i = k$, by definition of $M$ this implies that $c_0^b, c_k^b$ are the starting and ending configurations.

**Step 3b: connecting configurations.** It suffices to show that for every $i\in[k]$,

$$\Pr_{\mathsf{EXP}_{\mathsf{Ideal}}}\left[\ \forall b\in\{0,1\}.\Gamma(c_{i-1}^b, x_i^b, r_i^b) = c_i^b \ : \ \mathsf{Sim}(1^\lambda)\neq\perp\ \right] \geq 1 - \frac{1}{8kP(\Lambda)}\,.$$

Let $i\in[k]$. Consider $Y'_i, Y_{i+1}$, independent uniform distributions over $L'_i, L_{i+1}$. By Lemma 6.3, it suffices to show that

$$\Pr_{\mathsf{EXP}_{\mathsf{Ideal}}}\left[\ \mathrm{SD}\left(f'_i(Y'_i), f_{i+1}(Y_{i+1})\right) \geq \tfrac{1}{9kP(\Lambda)} \ : \ \mathsf{Sim}(1^\lambda)\neq\perp\ \right] \leq \mathsf{negl}(\Lambda)\,.$$

We recall that $L_j$ is a list whose elements are sampled by the experiment $\mathsf{EXP}_j$ defined as follows:

- Generate $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, j)$. Parse $\mathsf{dk} = \mathsf{td}_{\mathsf{seBARG}}$.
- Run the adversary and obtain $(X^0, X^1, a^0, a^1) \leftarrow \mathcal{A}(\mathsf{vk}, q)$.
- If $\mathsf{Verify}(\mathsf{vk}, \Gamma^0, X^0, a^0) = 0$ or $\mathsf{Verify}(\mathsf{vk}, \Gamma^1, X^1, a^1) = 0$, output $\perp$.

- For $b \in \{0,1\}$, parse $X^b = (x_i^b)_{i\in[k]}$ and $a^b = (\mathrm{rt}^b, \pi^b_{\mathsf{seBARG}})$.
- For $b \in \{0,1\}$, extract $w_1^b, w_j^b \leftarrow \mathsf{Extract}_{\mathsf{seBARG}}(\mathrm{td}_{\mathsf{seBARG}}, \pi^b_{\mathsf{seBARG}})$.
- For $b \in \{0,1\}$, parse $w_j^b = (r_j^b, c_{j-1}^b, c_j^b, \rho_{j-1}^b, \rho_j^b)$.
- Output $(c_{j-1}^b, x_j^b, r_j^b)_{b\in\{0,1\}}$.

In fact, each element in $L_j$ is distributed like a random sample from $\mathsf{EXP}_j$ conditioned on it being different than $\perp$. Let $\tilde{Y}_i, \tilde{Y}_{i+1}$ be independent uniform distributions over $\mathsf{EXP}_i, \mathsf{EXP}_{i+1}$ conditioned on not $\perp$. Since $f_i'$ only depends on the last element of its tuple, we abuse notation and use $f_i'(\tilde{Y}_i)$.

We prove the following claims:

**Claim 6.4.** *There exists a negligible function* $\mathsf{negl}$ *such that for every* $i \in [k]$,

$$\mathrm{SD}\left(f_i'(\tilde{Y}_i), f_{i+1}(\tilde{Y}_{i+1})\right) \le \mathsf{negl}(\Lambda) .$$

**Claim 6.5** ( [BDKP22]). *Let $D$ be a random variable supported on a set $X$. For every $\delta > 0$ and $n \in \mathbb{N}$ let $\ell = \delta^{-2} \cdot |X|^2 \cdot n$ and let $L \in X^\ell$ be a list where each element is sampled independently from $D$. Let $U_L$ be the uniform distribution over $L$. Then we have that*

$$\Pr_L\left[\ \mathrm{SD}\left(D, U_L\right) \ge \delta\ \right] \le 2^{-n} \cdot |X| .$$

We observe that the two claims imply that

$$\Pr_{\mathsf{EXP}_{\mathsf{Ideal}}}\left[\ \mathrm{SD}\left(f_i'(Y_i'), f_{i+1}(Y_{i+1})\right) \ge \tfrac{1}{9kP(\Lambda)} \quad : \quad \mathsf{Sim}(1^\lambda) \neq \perp\ \right] \le \mathsf{negl}(\Lambda) .$$

Indeed, considering $D = f_i'(\tilde{Y}_i)$, since it is supported on a set of size $2^{2S}$ and by our definition of $\ell$, by Claim 6.5 we have that $f_i'(\tilde{Y}_i), f_i'(Y_i)$ are $\frac{1}{20kP(\Lambda)}$-close except with negligible probability. Since $L_i'$ is a coupling of $L_{i-1}'$ and $L_i$, by Lemma 6.3 we have that the last element in $Y_i'$ is identically distributed to $Y_i$, so $f_i'(Y_i) \equiv f_i'(Y_i')$.

Similarly, for $Y_{i+1}$ we have that $f_{i+1}(\tilde{Y}_{i+1}), f_{i+1}(Y_{i+1})$ are $\frac{1}{20kP(\Lambda)}$-close except with negligible probability. Therefore, by a hybrid argument and Claim 6.4, we get our result.

We now prove the first claim (Claim 6.5 is taken from [BDKP22]).

**Proof of Claim 6.4.** We show that $f_i'(\tilde{Y}_i) \approx f_{i+1}(\tilde{Y}_{i+1})$, where $\approx$ denotes that the distributions are indistinguishable to poly($\Lambda$)-size distinguishers. This suffices, since the distributions are over a support of size $2S = O(\log \Lambda)$, so we get that the statistical distance is also negligible.

We first introduce new notation. We define alternative experiments $\mathsf{EXP}_{j_0,j_1,b'}$, identical to $\mathsf{EXP}_j$ except for the following changes:

- We generate $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, \{j_0, j_1\})$ rather than by $\mathsf{Query}(1^\lambda, U, 1^k, 1^n, 1^w, j)$, where in a query for a set we program the seBARG on $I = \{j_0, j_1\}$ rather than on $\{1, j\}$.
- For $b \in \{0,1\}$ we extract both $w_{j_0}^b, w_{j_1}^b \leftarrow \mathsf{Extract}_{\mathsf{seBARG}}(\mathrm{td}_{\mathsf{seBARG}}, \pi^b_{\mathsf{seBARG}})$, and continue parsing $w_{j_{b'}}^b$.

With this definition, $\mathsf{EXP}_j$ (from which $\tilde{Y}_j$ is sampled) is identical to $\mathsf{EXP}_{1,j,1}$, and we have

$$\begin{aligned}\tilde{Y}_i &= (c_{i-1}^b, x_i^b, r_i^b)_{b\in\{0,1\}}\,,\\ f_i'(\tilde{Y}_i) &= (\Gamma(c_{i-1}^b, x_i^b, r_i^b))_{b\in\{0,1\}}\,,\\ \tilde{Y}_{i+1} &= (c_i^b, x_{i+1}^b, r_{i+1}^b)_{b\in\{0,1\}}\,,\\ f_{i+1}(\tilde{Y}_{i+1}) &= (c_i^b)_{b\in\{0,1\}}\,.\end{aligned}$$

We proceed via a hybrid argument. First, by seBARG somewhere argument of knowledge and since we conditioned on $\forall b \in \{0,1\}.\mathsf{Verify}(\mathsf{vk}, \Gamma^b, X^b, a^b) = 1$ we have that in $\mathsf{EXP}_{1,i,1}$, except with negligible probability,

$$(\Gamma(c_{i-1}^0, x_i^0, r_i^0), \Gamma(c_{i-1}^1, x_i^1, r_i^1)) = (c_i^0, c_i^1)\,,$$

so

$$f_i'(\tilde{Y}_i) \approx \{c_i^0, c_i^1\}_{\mathsf{EXP}_{1,i,1}}\,.$$

Now, by seBARG index hiding we have

$$\{c_i^0, c_i^1\}_{\mathsf{EXP}_{1,i,1}} \approx \{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i,1}}\,.$$

Now, by seBARG somewhere argument of knowledge and by the collision resistance wrt opening of the hash, in $\mathsf{EXP}_{i,i}$ since for $b \in \{0,1\}$ both $w_0^b$ and $w_1^b$ contain valid openings of $\mathsf{rt}^b$ to some $c_i^b$ at index $i$, they must be equal except with negligible probability. Therefore,

$$\{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i,1}} \approx \{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i,0}}\,.$$

Now, by seBARG index hiding,

$$\{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i,0}} \approx \{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i+1,0}}\,.$$

Again by seBARG somewhere argument of knowledge and by the collision resistance wrt opening of the hash,

$$\{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i+1,0}} \approx \{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i+1,1}}\,.$$

Finally, we again use seBARG index hiding, and get that

$$\{c_i^0, c_i^1\}_{\mathsf{EXP}_{i,i+1,1}} \approx \{c_i^0, c_i^1\}_{\mathsf{EXP}_{1,i+1,1}} \equiv f_{i+1}(\tilde{Y}_{i+1})\,,$$

which finishes the proof of Claim 6.4.

**Step 4: Conclude Real and Ideal are indistinguishable.** We recall that it suffices to bound Eq. (3). In fact, since $E_\perp$ occurs with the same probability (up to $\mathsf{negl}(\Lambda)$) in $\mathsf{EXP}_{\mathsf{Real}}$ and $\mathsf{EXP}_{\mathsf{Ideal}}$, and since we've shown in step 2 that in $\mathsf{EXP}_{\mathsf{Ideal}}$ conditioned on $\neg E_\perp$ we have $\mathsf{Sim}(1^\lambda) \neq \perp$ except with negligible probability, it suffices to show

$$\left|\Pr_{\mathsf{EXP}_{\mathsf{Real}}}\left[\ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda,t)) = 1 \quad : \quad \neg E_\perp\ \right] - \Pr_{\mathsf{EXP}_{\mathsf{Ideal}}}\left[\ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda,t)) = 1 \quad : \quad \mathsf{Sim}(1^\lambda) \neq \perp\ \right]\right| < \frac{1}{2P(\Lambda)}\,.$$

We consider $(x_t^0, x_t^1, r_t^0, r_t^1)$ output by $\mathsf{Real}_{\mathcal{A}}(\lambda, t)$ conditioned on $\neg E_\perp$. By our definition of Sim, it is identical to the distribution of elements added to $S_t$. Therefore, by Lemma 6.3, we have that a random sample $(c_{i-1}^b, x_i^b, r_i^b)_{i\in[k],b\in\{0,1\}} \leftarrow S_k'$ is marginally identically distributed to $(x_t^0, x_t^1, r_t^0, r_t^1)$.

Thus, conditioned on $\mathsf{Sim}(1^\lambda) \neq \bot$, and considering $X^0, X^1, D^0, D^1 \leftarrow \mathsf{Sim}(1^\lambda)$, we have that $X^0[t], X^1[t]$, $D^0[t], D^1[t]$ are identically distributed to $x_t^0, x_t^1, r_t^0, r_t^1 \leftarrow \mathsf{Real}_{\mathcal{A}}(\lambda, t)$.

Now, we observe that the output of $\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, t)$ is exactly $X^0[t], X^1[t], D^0[t], D^1[t]$, except when Sim outputs $X^0, X^1, D^0, D^1$ that don't satisfy the policy. However, we showed in step 3 that conditioned on $\mathsf{Sim}(1^\lambda) \neq \bot$ in $\mathsf{EXP}_{\mathsf{Ideal}}$ this only occurs with probability $\leq \frac{1}{6P(\Lambda)}$, which concludes the proof of Theorem 6.2.

**Remark 6.2.** Finally, we note that the above construction and analysis extend to any $p$-composable vPIR for $p = O(1)$ (or, more generally, to $p \cdot S = O(\log \Lambda)$ assuming $\Lambda$-security of the underlying building blocks, where $S$ is the space bound of the constraint $\Gamma$).

# 7 BARGs with Adaptive Subset Extraction for Bounded-Space Policies

In this section, we use our constructed 2-composable vPIR for policies (Definition 6.1) to construct batch arguments with adaptive subset extraction.

We proceed by defining the families of policies we support and stating our main theorems, which we prove in Section 7.1 and Section 7.2.

**Definition 7.1** (Read-Once Space-$S$ Policies)**.** *Let $S(\lambda)$ be a function and let $F = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of functions. $F$ is a read-once space-$S$ family of policies if there exist polynomials $T(\lambda), N(\lambda), k(\lambda)$ such that each $f \in F_\lambda$ is a monotone function $f\colon \{0,1\}^k \to \{0,1\}$, and there exists a Turing machine $\Gamma \in \{0,1\}^N$ such that $f(b_1, \ldots, b_k) = 1$ if and only if for every $i \in [k-1]$ there exists $c_i \in \{0,1\}^S$ such that $\Gamma(c_{i-1}, b_i)$ outputs $c_i$ within $T$ steps where $c_0, c_k$ are some fixed starting and accepting configurations.*

**Theorem 7.2** (Section 7.1)**.** *Let $\Lambda = \Lambda(\lambda)$ be a function. Assuming a $\Lambda$-secure 2-composable* vPIR *scheme for policies, there exists a scheme* $\mathsf{BARG} = (\mathsf{Setup}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *such that for any family $F$ of read-once space $S$ policies where $S = O(\log \Lambda)$,* BARG *is a $F$-batch argument with $O(k)$-adaptive subset extraction for relation $\mathcal{R}_{\mathsf{full}}$.*

Using Theorem 5.4 and Theorem 7.2 above, we obtain the following theorem on aggregate signatures.

**Theorem 7.3.** *Let $\Lambda = \Lambda(\lambda)$ be a function. Assuming a $\Lambda$-secure 2-composable* vPIR *scheme for policies and a $\Lambda$-secure hash family with local opening, there exists a scheme* $\mathsf{AggS} = (\mathsf{Setup}, \mathsf{KeyAgg}, \mathsf{SigAgg}, \mathsf{AggVerify})$ *such that for any family $F$ of read-once space $S$ policies where $S = O(\log \Lambda)$ and for any digital signature scheme* S*,* AggS *is a $F$-aggregation scheme for* S*.*

We define below a specific family of monotone function, that of threshold policies.

**Definition 7.4** (Weighted Threshold Policies)**.** *Let $F = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of functions. $F$ is a family of weighted threshold policies if there exist polynomials $k(\lambda), \{\alpha_i(\lambda)\}_{i \in [k]}, t(\lambda)$ such that each $f \in F_\lambda$ is a function $f\colon \{0,1\}^k \to \{0,1\}$ such that $f(b_1, \ldots, b_k) = 1$ if and only if $b_i \in \{0,1\}$ and $\sum_{i=1}^k \alpha_i b_i \geq t$.*

We prove in Section 7.2 the following theorem for threshold policies.

**Theorem 7.5** (Section 7.2)**.** *Assuming a (polynomially-secure) 2-composable* vPIR *scheme for policies, there exists a scheme* $\mathsf{BARG} = (\mathsf{Setup}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *such that for any family $F$ of weighted threshold policies with threshold $t$,* BARG *is a $F$-batch argument with $O(t)$-adaptive subset extraction for relation $\mathcal{R}_{\mathsf{sub}}$ (where the aggregation time is polynomial in $t$).*

Thus, again using Theorem 5.4 and the Theorem 7.5 we have the following theorem.

**Theorem 7.6.** *Assuming a 2-composable* vPIR *scheme for policies and a hash family with local opening, there exists a scheme* AggS = (Setup, KeyAgg, SigAgg, AggVerify) *such that for any family $F$ of weighted threshold policies with threshold $t$ and for any digital signature scheme* S, AggS *is a $F$-aggregation scheme for* S *with aggregation time polynomial in $t$.*

Note that by setting $\alpha_i = 1$, in the above theorems, we revert to the (unweighted) threshold setting, where the aggregation time now depends on the number of signatures $t$.

## 7.1 Adaptive Subset Extraction for Bounded-Space Policies

In this section we prove Theorem 7.2. We begin by describing our construction, then prove its security.

### 7.1.1 Construction.

Our construction uses as a building block a $\Lambda$-secure 2-composable vPIR scheme for policies

$$(\mathsf{Query}, \mathsf{Answer}, \mathsf{Dec}, \mathsf{Verify}) ,$$

where we use the syntax for index instances (as in Remark 6.1).

We now describe the batch argument algorithms in Fig. 4.

### 7.1.2 Analysis.

We now prove security of our construction in Section 7.1.1, and obtain Theorem 7.2.

In the analysis, we sometimes abuse notation and omit the instances $X$ from the Answer, Verify algorithms and simulation security experiments, in case that $X = \perp$ (which means that our policy is just a standard vPIR constraint, without instances).

**Proof of Theorem 7.2.**

**Completeness.** Follows directly from the completeness of the 2-composable vPIR scheme for policies.

**Succinctness.** By the vPIR efficiency we have $|\pi| \leq m \cdot \mathsf{poly}(\lambda)$.

**$O(k)$-Adaptive Subset Extraction.** Let $S = O(\log \Lambda)$ and let $F$ be a family of read-once space-$S$ policies. Fix any polynomial $T(\lambda)$, any poly-size cheating prover $\mathcal{P}^*$, and any sequence $\{f_\lambda \in F_\lambda\}_{\lambda \in \mathbb{N}}$. Let EXP be the experiment defined in the adaptive subset extraction requirement:

- Generate $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- Run the cheating prover and obtain $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
- Let $x = (f_\lambda, M, z, T)$.
- Extract $(i, w_i) \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)$.

We define an adversary $\mathcal{A}$ against the $\Lambda$-security of the 2-composable vPIR scheme for policies. Given $\mathsf{vk}, q$, the adversary $\mathcal{A}$ does the following:

$\mathsf{Setup}(1^\lambda)$:

1. For every $j \in [\lambda]$, sample $i_j \leftarrow [2^j]$ uniformly at random.
2. For every $j \in [\lambda]$, generate $(\mathsf{dk}_j, \mathsf{vk}_j, q_j) \leftarrow \mathsf{Query}(1^\lambda, i_j)$.
3. Output $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]}$, $\mathsf{td} = (i_j, \mathsf{dk}_j)_{j\in[\lambda]}$.

$\mathcal{P}(\mathsf{crs}, f, M, z, 1^T, w_1, \ldots, w_k)$:

1. Parse $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]}$.
2. Let $(q, \mathsf{vk}) = (q_j, \mathsf{vk}_j)$ for $j = \lceil \log k \rceil$ (where recall that $f\colon \{0,1\}^k \to \{0,1\}$).
3. Let $\Gamma_{f,T}$ be the policy-checking constraint defined by $\Gamma_f$ and $T$: given $c_{i-1}, x_i, r_i$ as input, $\Gamma_{f,T}$ does the following:
   (a) Parse $x_i = (M, z, i)$ and $r_i = w_i$.
   (b) Let $b_i = 1$ if $M(z, i, w_i)$ accepts within $T$ steps, and $b_i = 0$ otherwise.
   (c) Output $c_i = \Gamma_f(c_{i-1}, b_i)$.
4. Let $D = (w_i)_{i\in[k]}$.
5. Output $\pi = \mathsf{Answer}(D, \Gamma_{f,T}, (M, z), q)$.

$\mathcal{V}(\mathsf{crs}, x, \pi)$:

1. Parse $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]}$, $x = (f, M, z, T)$ and $\pi = a$.
2. Let $(q, \mathsf{vk}) = (q_j, \mathsf{vk}_j)$ for $j = \lceil \log k \rceil$.
3. Let $\Gamma_{f,T}$ as in $\mathcal{P}$.
4. Output $\mathsf{Verify}(\mathsf{vk}, \Gamma_{f,T}, (M, z), a)$.

$\mathsf{Extract}(\mathsf{td}, \pi)$:

1. Parse $\mathsf{td} = (i_j, \mathsf{dk}_j)_{j\in[\lambda]}$ and $\pi = a$.
2. Let $(i, \mathsf{dk}) = (i_j, \mathsf{dk}_j)$ for $j = \lceil \log k \rceil$.
3. Extract $w_i = \mathsf{Dec}(\mathsf{dk}, a)$.
4. Output $(i, w_i)$.

Figure 4: Construction of BARGs with Adaptive Subset Extraction for bounded-space policies

1. Sample $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]} \leftarrow \mathsf{Setup}(1^\lambda)$ and replace $(q_j, \mathsf{vk}_j)$ for $j = \lceil \log k \rceil$ with $(q, \mathsf{vk})$.
2. Run the cheating prover and obtain $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
3. Let $f = f_\lambda$, and $\Gamma_{\bar{f}}$ be a constraint such that $\Gamma_{\bar{f}}$ is accepting iff $\Gamma_f$ is rejecting.
4. Let $D_{\bar{J}} = \mathbb{1}_{[k]\setminus J} = \left(\bar{J}_i\right)_{i\in[k]}$.
5. Compute $a' = \mathsf{Answer}(D_{\bar{J}}, \Gamma_{\bar{f}}, q)$.
6. Output $((M, z, i)_{i\in[k]}, \bot, a, a')$ where $a = \pi$.

Let $P$ be a polynomial. By 2-composable security of the underlying vPIR scheme, there exists a poly($\Lambda$)-size simulator Sim such that for any poly($\Lambda$)-size distinguisher $\mathcal{D}$ and for any $\lambda \in \mathbb{N}$ and row index $t \in [k]$,

$$\left|\Pr\left[\ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, t)) = 1\ \right] - \Pr\left[\ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, t)) = 1\ \right]\right| < \frac{1}{P(\Lambda)}\ .$$

First, we observe that in the real experiment, for every $\lambda \in \mathbb{N}$ let $j = \lceil \log k \rceil$ then we have

$$\Pr\left[\ \mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, t) \neq \bot \quad : \quad t \leftarrow [2^j]\ \right] \geq \Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right],$$

since $\mathsf{Real}_{\mathcal{A}}$ with a random $t \leftarrow [2^j]$ runs $\mathcal{P}^*$ with an identical input distribution to the one in EXP, and we have that (i) $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ implies that $\mathsf{Verify}(\mathsf{vk}, \Gamma_{f,T}, (M, z), a) = 1$, and (ii) $f(\mathbb{1}_{[k]\setminus J}) = 0$ implies that $D_{\bar{J}}$ defined in $\mathcal{A}$ satisfies the constraint $\Gamma_{\bar{f}}$, which by completeness of vPIR implies $\mathsf{Verify}(\mathsf{vk}, \Gamma_{\bar{f}}, a') = 1$. Combining (i) and (ii) we get that $\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, t) \neq \bot$.

We now use 2-composable simulation security to switch to the ideal experiment. Let $t \in [2^j]$ such that the above inequality holds (exists by averaging), and let $T'$ be the time bound for the constraints $\Gamma_{f,T}, \Gamma_{\bar{f}}$ then

$$\begin{aligned} &\Pr\left[\begin{array}{ll} U_{T',S}(\Gamma_{f,T}, X, D) = 1 \wedge & : \quad (X, \bot, D, D') \leftarrow \mathsf{Sim}(1^\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}) \\ U_{T',S}(\Gamma_{\bar{f}}, D') = 1 & \end{array}\right] \\ &= \Pr\left[\ \mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, t) \neq \bot\ \right] > \Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] - \frac{1}{P(\Lambda)}\ . \end{aligned}$$

By definition of $\Gamma_{f,T}$, we have that if $U_{T',S}(\Gamma_{f,T}, X, D) = 1$ then, parsing $X = (M_i, x_i)_{i\in[k]}$ and $D = (w_i)_{i\in[k]}$, it holds that for $b_i = M_i(x_i, w_i)$ we have $f(b_1, \ldots, b_k) = 1$. Similarly, if $U_{T',S}(\Gamma_{\bar{f}}, D') = 1$ then defining $J = \{i \in [k]\ :\ D'[i] = 0\}$ we have $f(\mathbb{1}_{[k]\setminus J}) = 0$.

Now, if $f(b_1, \ldots, b_k) = 1$ and $f(\mathbb{1}_{[k]\setminus J}) = 0$ (in other words, $b_1, \ldots, b_k$ is a satisfying assignment and $J$ is a necessary subset), since $f$ is a monotone function there must exist an $i \in [k]$ such that $b_i = 1$ and $i \in J$. Therefore, sampling an independent $i \leftarrow [k]$ in the above experiment, we get that

$$\begin{aligned} &\Pr\left[\ \bar{J}_i = 0\ \wedge\ M_i(x_i, w_i) = 1 \quad : \quad \begin{array}{l} i \leftarrow [k] \\ ((M_i, x_i), \bot, w_i, \bar{J}_i) \leftarrow \mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, i) \end{array}\right] \\ &> \frac{1}{k(\lambda)}\left(\Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] - \frac{1}{P(\Lambda)}\right)\ . \end{aligned}$$

Switching to the $\mathsf{Real}_{\mathcal{A}}$ experiment and using 2-composable simulation security, we get that

$$\begin{aligned} &\Pr\left[\ \bar{J}_i = 0\ \wedge\ M(z, i, w_i) = 1 \quad : \quad \begin{array}{l} i \leftarrow [k] \\ ((M, z, i), \bot, w_i, \bar{J}_i) \leftarrow \mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, i) \end{array}\right] \\ &> \frac{1}{k(\lambda)}\left(\Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] - \frac{1}{P(\Lambda)}\right) - \frac{1}{P(\Lambda)}\ . \end{aligned}$$

We now observe that the experiment of running $\mathcal{A}$ above is identical to running $\mathcal{P}^*$ in EXP, conditioned on $i_j \in [k]$ for $j = \lceil \log k \rceil$ (which occurs with probability $\geq \frac{1}{2}$ by choice of $j$). In particular, we have that (i) $\mathsf{Dec}(\mathsf{dk}, a) = w_i$ above is identical to $w_i = \mathsf{Extract}(\mathsf{td}, \pi)$ in EXP, and (ii) by vPIR completeness and

construction of $\mathcal{A}$, we have that $\mathsf{Dec}(\mathsf{dk}, a') = \bar{J}_i$ corresponds to $i \notin J$ where $J$ is the set given by $\mathcal{P}^*$ in EXP. Therefore, we have

$$\begin{aligned}
&\Pr_{\mathsf{EXP}}\left[\ i \in J \ \wedge\ M(z, i, w_i) = 1\ \right] \\
&\geq \frac{1}{2}\Pr\left[\ \bar{J}_i = 0 \ \wedge\ M(z, i, w_i) = 1 \quad : \quad \begin{array}{l} i \leftarrow [k] \\ ((M, z, i)_{i\in[k]}, \bot, w_i, \bar{J}_i) \leftarrow \mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, i) \end{array}\ \right] \\
&> \frac{1}{2k(\lambda)}\left(\Pr_{\mathsf{EXP}}\left[\ \begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\ \right] - \frac{1}{P(\Lambda)}\right) - \frac{1}{P(\Lambda)}\,.
\end{aligned}$$

Since this holds for any polynomial $P$, we get that there exists a negligible function $\mathsf{negl}$ such that

$$\Pr_{\mathsf{EXP}}\left[\ i \in J \ \wedge\ M(z, i, w_i) = 1\ \right] > \frac{1}{2k(\lambda)} \cdot \Pr_{\mathsf{EXP}}\left[\ \begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\ \right] - \mathsf{negl}(\lambda)\,,$$

which proves the $O(k)$-adaptive subset extraction property and concludes the proof of Theorem 7.2.

## 7.2 Adaptive Subset Extraction with Sublinear Prover for Threshold Policies

In this section, we use 2-composable vPIR to construct batch arguments with adaptive subset extraction for the sublinear relation $\mathcal{R}_{\mathsf{sub}}$. We fully describe a solution for the case of threshold policies, and sketch how to extend it to support general read-once bounded-space policies.

We begin by describing a general approach towards achieving a sublinear prover for any read-once bounded-space policy. The idea is to consider a compressed vPIR database, corresponding to the $\mathcal{R}_{\mathsf{sub}}$-witness: suppose the witness is $(i_1, w_1, \ldots, i_t, w_t)$, where $i_1, \ldots, i_t$ are sorted in increasing order. We consider the vPIR database that contains $t$ rows, each containing an index-witness pair. Then, the BARG verifier would verify that this database satisfies the constraint corresponding to the validity of this witness: (i) the indices are sorted, (ii) all witnesses are valid, and (iii) the input $b_1, \ldots, b_k$ where $b_i = 1 \iff \exists j$ s.t. $i = i_j$ satisfies the policy $f$.

First, we observe that for general bounded-space policies, it is not clear that the prover can evaluate this constraint on the database in sublinear time (i.e. in time that only depends on the compressed database size and not on $k$). Indeed, the difficulty lies in (iii) above: even if the policy $f$ is read-once bounded-space, to update the state between rows $j$ and $j+1$, the prover needs to update the state using the next $i_{j+1} - i_j$ input bits: $b_{i_j} = 1$ and $b_i = 0$ for $i_j < i < i_{j+1}$. This could take up to linear time. We propose two ways to handle this issue:

- First, we can restrict the class of policies $f$ to ones with a "fast-forward" property, where one can update $f$'s state reading any number of repeated zeroes in time that depends sublinearly in this number. It class includes, in particular, threshold and weighted threshold policies where reading zeroes does not change the state at all.

- Alternatively, we can also support general monotone policies, at the expense of having a long crs (exponential in the space), which the prover has random access to. This crs could contain a table of states of $f$ after any number of zeroes, which the prover can later use instead of performing the computation during proof generation (we note that in order to convince the verifier, the long crs should also contain SNARG proofs that each table entry is correct).

**Security for Threshold policies.** To argue security, we start by looking at the case of threshold policies. Similarly to our construction in Section 7.1, given a cheating prover that outputs a vPIR answer $a$ and a necessary subset $J$, we consider a composed prover that gives an additional vPIR answer based on the database corresponding to this subset $J$, and the constraint that checks that it is necessary, i.e. that $|J| > k - t$. To use composition, we encode $J$ as a database $D'$ of size $t$ (matching the size of the prover's database), consisting of rows $(\tilde{i}_{j-1}, \tilde{i}_j)$ where $\tilde{i}_j$ are such that the edges are $\tilde{i}_0 = 0$ and $\tilde{i}_t = k + 1$, and the points cover $\bar{J}$, i.e. $\bar{J} \subseteq \{\tilde{i}_1, \ldots, \tilde{i}_{t-1}\}$.

Given this cheating prover, our composition theorem guarantees a simulator for the joint distribution of the databases $D = (i_j, w_j)_{j \in [t]}$ and $D' = (\tilde{i}_{j-1}, \tilde{i}_j)_{j \in [t]}$. Now, we use the combinatorial property that every pair of valid databases $D, D'$ must contain a row $j$ such that $i_j$ is contained in the open interval $(\tilde{i}_{j-1}, \tilde{i}_j)$, which implies that $i_j \in J$. This gives us the adaptive subset extraction property: sampling a random row in the database, with probability $\frac{1}{k}$ we're guaranteed to find a row $j$ such that $w_j$ is valid and $i_j \in J$.

This analysis also extends to weighted threshold policies, by expanding the database to contain multiple copies of each witness, corresponding to the weight. The full analysis for the case of weighted threshold is given below. Before describing this analysis, we explain why the same ideas cannot be used to argue security for more general policies and sketch an alternative argument for this case.

**Security for general policies.** Unfortunately, it seems that the same approach doesn't extend to general read-once bounded-space policies. Instead, we can argue security using a direct analysis, utilizing additional properties of the vPIR construction in Section 6 that do not seem to be captured by our composition theorem.

In a nutshell, the issue is with defining the encoding of $J$ as a database. For the case of thresholds, we exploited the fact that the complement of a necessary subset must be smaller in size than any satisfying subset to the policy, to encode $J$ as a sequence of $t$ open intervals in $J$. For general policies, where this fact does not hold, it's not clear how to encode $J$. We could try to compress $\bar{J}$ to $t$ rows by encoding multiple such intervals in each row, but for any fixed choice of the number of intervals in each row, we could find a set of valid databases where our combinatorial property doesn't hold, i.e. there is no row $j$ such that the index $i_j$ is contained in some open interval in the same row.

To solve this problem, we can open up the proof of our composition theorem, and define a "dynamic" database structure for $J$ that depends on the rows $(i_j, w_j)$ in $D$ to choose precisely how many elements of $\bar{J}$ to include in each row. In more detail, using the same simulation strategy as in Section 6 we can repeatedly query $\mathcal{P}^*$ on each index $j$, by running $\mathcal{P}^*(q) \to (a, J)$ for $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, j)$ and decrypting the database row $(i_{j-1}, i_j, w_j) = r_j = \mathsf{Dec}(\mathsf{dk}, a)$ (we include the previous index $i_{j-1}$ as part of each database row). Then, we construct a list $L_j$ of answers of the form $(i_{j-1}, i_j, w_j, \bar{J} \cap (i_{j-1}, i_j])$. Finally, using the coupling lemma Lemma 6.3, we can couple together the lists $L_1, \ldots, L_t$ according to $(i_{j-1}, i_j, w_j)_{j \in [t]}$ being an accepting database (i.e., $i_j$ that appears in the $j$th and $j+1$-th rows should be equal, and $w_j$ should all be valid witnesses), and the set $\bar{J}$ composed of the intervals $\bar{J} \cap (i_{j-1}, i_j]$ not satisfying the policy, i.e. $J$ being a necessary subset (note these two conditions can be checked with a bounded size state between rows).

We then have that in any such pair of databases, since the set of witnesses satisfy the policy and $J$ is a necessary subset, there must exist an index $j \in J$ such that $w_j$ is an accepting witness for the $i_j$th statement, and $i_j \in J$. But now, since we defined $J$'s database structure such that its rows $J \cap (i_{j-1}, i_j]$ exactly match the indices of valid witnesses, we have that just row $j$ in the simulated pair of databases is enough to decide if $J$ contains the index $i_j$. Thus this property is maintained in the real world experiment, and we get $O(t)$-adaptive subset extraction by extracting at a random row index $j$.

In what follows we describe the analysis for the case of weighted threshold.

### 7.2.1 Construction.

Our construction uses as a building block a composable polynomially-secure vPIR scheme for policies

$$(\mathsf{Query}, \mathsf{Answer}, \mathsf{Dec}, \mathsf{Verify}) ,$$

where we use the syntax for index instances (as in Remark 6.1).

### 7.2.2 Analysis.

We now prove security of our construction in Section 7.2, and obtain Theorem 7.5.

**Proof of Theorem 7.5.**

**Completeness.** Follows directly from the completeness of the 2-composable vPIR scheme for policies.

**Succinctness.** By the vPIR efficiency we have $|\pi| \leq m \cdot \mathsf{poly}(\lambda)$.

**$O(t)$-Adaptive Subset Extraction.** Let $F$ be a family of weighted threshold policies. Fix any polynomial $T(\lambda)$, any poly-size cheating prover $\mathcal{P}^*$, and any sequence $\{f_\lambda \in F_\lambda\}_{\lambda\in\mathbb{N}}$. Let $\mathsf{EXP}$ be the experiment defined in the adaptive subset extraction requirement:

- Generate $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- Run the cheating prover and obtain $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
- Let $x = (f_\lambda, M, z, T)$.
- Extract $(i, w_i) \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)$.

We define an adversary $\mathcal{A}$ against the $\Lambda$-security of the 2-composable vPIR scheme for policies. Given $\mathsf{vk}, q$, the adversary $\mathcal{A}$ does the following:

1. Suppose that $f = f_\lambda$ is a weighted threshold function with $k$ inputs, weights $\{\alpha_i\}_{i\in[k]}$ and threshold $t$.
2. Sample $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]} \leftarrow \mathsf{Setup}(1^\lambda)$ and replace $(q_j, \mathsf{vk}_j)$ for $j = \lceil \log t \rceil$ with $(q, \mathsf{vk})$.
3. Run the cheating prover and obtain $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
4. Define the constraint $\Gamma_{\tilde{f}}$ which does the following given $c_{j-1}, r_j$ as input:
   (a) Parse the state $c_{j-1}$. The state is either $c_{j-1} = \perp$, or the starting state $c_{j-1} = c_0$, or a tuple $c_{j-1} = (a, b, c)$ where $a < b \in [k] \cup \{-\infty, \infty\}$ are indices, and $c \in [\alpha_b]$ is a counter (where $\alpha_\infty = \infty$).
   (b) Parse the row $r_j = (a', b')$.
   (c) Define the next state $c_j$ via one of the following cases:
   - If $c_{j-1} = c_0$ and $-\infty = a' < b' \in [k] \cup \{\infty\}$, set $c_j = (a', b', 1)$.
   - If $1 \leq c < \alpha_b$ and $a = a'$ and $b = b'$, set $c_j = (a, b, c + 1)$.
   - If $c = \alpha_b$ and $b = a' < b'$, set $c_j = (a', b', 1)$.

$\mathsf{Setup}(1^\lambda)$ does the following:

1. For every $j \in [\lambda]$, sample $i_j \leftarrow [2^j]$ uniformly at random.
2. For every $j \in [\lambda]$, generate $(\mathsf{dk}_j, \mathsf{vk}_j, q_j) \leftarrow \mathsf{Query}(1^\lambda, i_j)$.
3. Output $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]}, \mathsf{td} = (\mathsf{dk})_{j\in[\lambda]}$.

$\mathcal{P}(\mathsf{crs}, f, M, z, 1^T, \{i_j, w_j\}_{j\in[t]})$ does the following:

1. Parse $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]}$.
2. Suppose that $f$ is a weighted threshold function with $k$ inputs, weights $\{\alpha_i\}_{i\in[k]}$ and threshold $t$. We define the policy-checking constraint $\Gamma_{f,T}$ which does the following given $c_{j-1}, x_j, r_j$ as input:
   (a) Parse the state $c_{j-1}$. The state is either $c_{j-1} = \perp$, or the starting state $c_{j-1} = c_0$, or a tuple $c_{j-1} = (i, c)$ where $i \in [k]$ is an index, and $c \in [\alpha_i]$ is a counter.
   (b) Parse the instance $x_j = (M, z, j)$ and the row $r_j = (i', w_{i'})$.
   (c) Define the next state $c_j$ via one of the following cases:
      i. If $c_{j-1} = c_0$ and $1 \le i' \le k$ and $M(z, i', w_{i'})$ accepts within $T$ steps, set $c_j = (i', 1)$.
      ii. If $1 \le c < \alpha_i$ and $i = i'$, set $c_j = (i, c+1)$.
      iii. If $c = \alpha_i$ and $i < i' \le k$ and $M(z, i', w_{i'})$ accepts within $T$ steps, set $c_j = (i', 1)$.
      iv. Otherwise, set $c_j = \perp$.
   (d) If $j = t$ and $c_j \neq \perp$, output the accepting state. Otherwise, output $c_j$.
3. Let $(q, \mathsf{vk}) = (q_j, \mathsf{vk}_j)$ for $j = \lceil \log t \rceil$.
4. Let $D = (i_j, w_j)_{j\in[t]}$ (we assume $i_j$ are sorted).
5. Output $\pi = \mathsf{Answer}(D, \Gamma_{f,T}, (M, z), q)$.

$\mathcal{V}(\mathsf{crs}, x, \pi)$ does the following:

1. Parse $\mathsf{crs} = (q_j, \mathsf{vk}_j)_{j\in[\lambda]}, x = (f, M, z, T)$ and $\pi = a$.
2. Let $(q, \mathsf{vk}) = (q_j, \mathsf{vk}_j)$ for $j = \lceil \log t \rceil$.
3. Let $\Gamma_{f,T}$ as in $\mathcal{P}$.
4. Output $\mathsf{Verify}(\mathsf{vk}, \Gamma_{f,T}, (M, z), a)$.

$\mathsf{Extract}(\mathsf{td}, \pi)$ does the following:

1. Parse $\mathsf{td} = (\mathsf{dk}_j)_{j\in[\lambda]}$ and $\pi = a$.
2. Let $\mathsf{dk} = \mathsf{dk}_j$ for $j = \lceil \log t \rceil$.
3. Output $(i_j, w_j) = \mathsf{Dec}(\mathsf{dk}, a)$.

Figure 5: Construction of Sublinear Prover BARGs for weighted threshold policies

- Otherwise, set $c_j = \perp$.

(d) If $i = t$ and $b = \infty$, output the accepting state. Otherwise, output $c_i$.

5. Let $\bar{J}_1, \ldots, \bar{J}_\ell$ be the elements of $\bar{J} = [k] \setminus J$ sorted in increasing order.
6. If $\sum_{i=1}^{\ell} \alpha_{\bar{J}_i} \ge t$, output $\perp$.

7. Otherwise, let $D_{\bar{J}}$ be the database that contains $\alpha_{\bar{J}_1}$ copies of the row $(-\infty, \bar{J}_1)$, followed by $\alpha_{\bar{J}_{i+1}}$ copies of the row $(\bar{J}_i, \bar{J}_{i+1})$ for every $1 \le i \le \ell$, and finally $t - \sum_{i=1}^{\ell} \alpha_{\bar{J}_i}$ copies of the row $(\bar{J}_s, \infty)$.

8. Compute $a' = \mathsf{Answer}(D_{\bar{J}}, \Gamma_{\bar{f}}, q)$.

9. Output $((M, z, j)_{j\in[t]}, \bot, a, a')$ where $a = \pi$.

Let $P$ be a polynomial. Now, as in the proof of Theorem 7.2, using the definition of $\mathcal{A}$ and 2-composable simulation security, let $T', S$ be the time and space bounds for the constraints $\Gamma_{f,T}, \Gamma_{\bar{f}}$ then we have

$$\begin{aligned}
&\Pr\left[\begin{array}{ll} U_{T',S}(\Gamma_{f,T}, X, D) = 1 \wedge & : \quad (X, \bot, D, D') \leftarrow \mathsf{Sim}(1^\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}) \\ U_{T',S}(\Gamma_{\bar{f}}, D') = 1 & \end{array}\right] \\
&= \Pr\left[\ \mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, i) \neq \bot \quad : \quad i \leftarrow [t]\ \right] \\
&> \Pr\left[\ \mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, i) \neq \bot \quad : \quad i \leftarrow [t]\ \right] - \frac{1}{P(\Lambda)} \ge \Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] - \frac{1}{P(\Lambda)}\,.
\end{aligned}$$

By definition of $\Gamma_{f,T}$ and $\Gamma_{\bar{f}}$, we have that:

- If $U_{T',S}(\Gamma_{f,T}, X, D) = 1$ then, parsing $X = (M_j, z_j, \cdot)_{j\in[t]}$ and $D = (i_j, w_j)_{j\in[t]}$, we have that (i) $\{i_j\}_{j\in[t]}$ is a weakly increasing sequence of indices in $[k]$, (ii) each $i_j$ appears $\alpha_{i_j}$ times (or does not appear at all), and (iii) for each $j \in [t]$, the machine $M_j(z_j, i_j, w_j) = 1$ accepts within $T$ steps.
- If $U_{T',S}(\Gamma_{\bar{f}}, D') = 1$ then, parsing $D' = (a_j, b_j)_{j\in[t]}$, we have that (i) each row has $a_j < b_j \in [k] \cup \{-\infty, \infty\}$, (ii) consecutive rows in $D'$ are either identical or such that $b_j = a_{j+1}$, (iii) $a_0 = -\infty$ and $b_t = \infty$, and (iv) for each $b_j \in [k]$ that appears in the database, we have exactly $\alpha_{b_j}$ copies of it.

Combining the above, we claim that there must exist an $j \in [t]$ such that $a_j < i_j < b_j$. Indeed, if this property does not hold for any $i \le t-1$, then since $-\infty = a_1 < i_1$ we must have $b_1 \le i_1$, and so by induction we argue that we always have $a_j < i_j$, and finally this implies that $a_t < i_t < \infty = b_t$ (so the property holds for $i = t$).

Therefore, we have that

$$\begin{aligned}
&\Pr\left[\begin{array}{ll} a_j < i_j < b_j \wedge & : \quad j \leftarrow [t] \\ M_j(z_j, i_j, w_j) = 1 & \quad ((M_j, z_j, \cdot), \bot, (i_j, w_j), (a_j, b_j)) \leftarrow \mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, j) \end{array}\right] \\
&\ge \frac{1}{t(\lambda)}\left(\Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] - \frac{1}{P(\Lambda)}\right)\,.
\end{aligned}$$

Switching to the $\mathsf{Real}_{\mathcal{A}}$ experiment and using 2-composable simulation security, we get that

$$\begin{aligned}
&\Pr\left[\begin{array}{ll} a_j < i_j < b_j \wedge & : \quad j \leftarrow [t] \\ M_j(z_j, i_j, w_j) = 1 & \quad ((M_j, z_j, \cdot), \bot, (i_j, w_j), (a_j, b_j)) \leftarrow \mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\bar{f}}, j) \end{array}\right] \\
&\ge \frac{1}{t(\lambda)}\left(\Pr_{\mathsf{EXP}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0 \end{array}\right] - \frac{1}{P(\Lambda)}\right) - \frac{1}{P(\Lambda)}\,.
\end{aligned}$$

Finally, as in Theorem 7.2, we observe that the experiment of running $\mathcal{A}$ above is identical to running $\mathcal{P}^*$ in EXP conditioned on $i_j \in [t]$ (which happens wp $> \frac{1}{2}$, and in particular:

- By definition, $\mathsf{Dec}(\mathsf{dk}, a)$ is identical to the extracted value $\mathsf{Extract}(\mathsf{td}, \pi) = (i_j, w_j)$ in EXP.

- By vPIR completeness and construction of $\mathcal{A}$, we have that $\mathsf{Dec}(\mathsf{dk}, a') = (a_j, b_j)$ contains indices such that any $a_j < i < b_j$ is contained in $J$ (since $D_{\bar{J}}$ is constructed to contain all elements in $\bar{J}$ in order).

Therefore, we have

$$\begin{aligned}
&\Pr_{\mathsf{EXP}}\left[\ i \in J \ \wedge\ M(z, i, w_i) = 1\ \right] \\
&\geq \frac{1}{2}\Pr\left[\begin{array}{ll} a_j < i_j < b_j \wedge & : \quad j \leftarrow [t] \\ M_j(z_j, i_j, w_j) = 1 & \quad ((M_j, z_j, \cdot), \bot, (i_j, w_j), (a_j, b_j)) \leftarrow \mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma_{f,T}, \Gamma_{\tilde{f}}, j)\end{array}\right] \\
&\geq \frac{1}{2t(\lambda)}\left(\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \frac{1}{P(\Lambda)}\right) - \frac{1}{P(\Lambda)}\,.
\end{aligned}$$

Since this holds for any polynomial $P$, we get that there exists a negligible function $\mathsf{negl}$ such that

$$\Pr_{\mathsf{EXP}}\left[\ i \in J \ \wedge\ M(z, i, w_i) = 1\ \right] > \frac{1}{2t(\lambda)} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mathsf{negl}(\lambda)\,,$$

which proves $O(t)$-adaptive subset extraction and concludes the proof of Theorem 7.5.

# 8 BARGs with Functional Subset Extraction for Monotone Circuit Policies

In this section, we construct functional subset extraction for monotone functions that can be represented as polynomial size circuits. We state the following theorem, and provide a proof sketch since analysis follows identically to the analysis presented in [BBK+23].

**Theorem 8.1.** *Assuming a somewhere-extractable batch argument scheme (Definition 3.5) and fully homomorphic encryption, there exists a scheme* $\mathsf{BARG} = (\mathsf{Setup}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *such that for any family $F$ of monotone circuit policies and family $G$ of functions computable by polynomial-size circuits,* BARG *is a $F$-batch argument with $G$-functional subset extraction for relation $\mathcal{R}_{\mathsf{full}}$ with hashed inputs.*

Combining Theorem 5.8 and Theorem 8.1 above, we have the following theorem about weakly unforgeable aggregate signatures.

**Theorem 8.2.** *Assuming a somewhere-extractable batch argument scheme (Definition 3.5) and fully homomorphic encryption, there exists a scheme* $\mathsf{AggS} = (\mathsf{Setup}, \mathsf{KeyAgg}, \mathsf{SigAgg}, \mathsf{AggVerify})$ *such that for any family $F$ of functions computable by polynomial-size monotone circuits and for any digital signature scheme* S *with trapdoor keys,* AggS *is a $F$-aggregation scheme for* S.

## 8.1 Construction and Proof Sketch

In this section we give a proof sketch for Theorem 8.1. We recall the construction of batch arguments for monotone circuit policies from [BBK+23], and show that it can support functional subset extraction (defined in Section 5.2).

In Section 8.1.1, we recall the notion of a predicate-extractable hash [BBK+23], and extend the definition to support functional predicate extraction. This extension was in fact already used in [BBK+23], to get a shorter common reference string for their SNARG.

Then, in Section 8.1.2 we recall the construction of a somewhere-extractable SNARG for monotone policy BatchNP from [BBK+23]. We show that instantiating this construction with a functional predicate-extractable hash, we get a batch argument with functional subset extraction.

### 8.1.1 Predicate Extractable Hash (PEHash)

In this section we define predicate-extractable hash with tags for bit-fixing predicates. The definition is taken from [BBK+23] (Definition 5.4). This is a notion that generalizes the notion of a somewhere-extractable hash [HW15, OPWW15]. Whereas a somewhere-extractable hash allows to generate a hash key $\mathsf{hk}_i$ programmed on an index $i$ and use a corresponding trapdoor $\mathsf{td}_i$ to extract $x_i$ from $\mathsf{rt} = \mathsf{Hash}(\mathsf{hk}_i, x)$, a predicate-extractable hash for a family $\mathcal{F}$ allows to generate a hash key programmed on a global predicate $f \in \mathcal{F}$, and extract the value of $f(x)$. [BBK+23] construct such hash families for the bit-fixing family of predicates, where each $f \in \mathcal{F}$ is defined by a set $J \subseteq [k]$ and a string $s \in \{0,1\}^J$, and we have $f(x) = 1$ iff $\forall j \in J,\ x_j = s_j$.

We consider an extended definition that supports functional predicate extraction (as in Remark 5.2 in [BBK+23]). This allows to program the PEHash on a function $g$ during key generation (rather than on a predicate specified by $J, s$), then specify an input $y$ to the function when hashing a string $x$. This produces a hash value, from which we can extract the value of the predicate $f = g(y)$ on $x$.

**Syntax.** Let $G = \{G_\lambda\}_{\lambda \in \mathbb{N}}$ be a collection of functions such that every $g \in G_\lambda$ maps inputs to bit-fixing predicates $f$. For $g \in G_\lambda$, we denote its description length as a circuit by $|g|$.

A $G$-functional predicate extractable hash family PEHash with tags with respect to the bit-fixing predicate family consists of the following polynomial-time algorithms:

$\mathsf{Gen}(1^\lambda, g) \to (\mathsf{hk}, \mathsf{vk}, \mathsf{td})$. This is a probabilistic setup algorithm that takes as input a security parameter $1^\lambda$ in unary, and a function $g \in G_\lambda$. It outputs a hash key hk, verification key vk and trapdoor td.

$\mathsf{Hash}(\mathsf{hk}, y, x, \vec{t}) \to \mathsf{v}$. This is a deterministic algorithm that takes as input a hash key hk, a function input $y$, a hash input $x \in \{0,1\}^N$, and tags $\vec{t} = (t_1, \ldots, t_N) \in (\{0,1\}^T)^N$. It outputs a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$.

$\mathsf{Open}(\mathsf{hk}, y, x, \vec{t}, j) \to \rho$. This is a deterministic algorithm that takes as input a hash key hk, a function input $y$, a hash input $x \in \{0,1\}^N$, tags $\vec{t} = (t_1, \ldots, t_N) \in (\{0,1\}^T)^N$ and an index $j \in [N]$. It outputs an opening $\rho \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$.

$\mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, b, t, \rho) \to 0/1$. This is a deterministic algorithm that takes as input a verification key vk, a function input $y$, a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, an index $j \in [N]$, a bit $b \in \{0,1\}$, a tag $t \in \{0,1\}^T$ and an opening $\rho \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs 1 (accept) or 0 (reject).

$\mathsf{Extract}(\mathsf{td}, y, \mathsf{v}) \to u$. This is a deterministic extraction algorithm that takes as input a trapdoor td, a function input $y$ and a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs a bit $u \in \{0,1\}$.

$\mathsf{ExtractIndex}(\mathsf{td}, y, \mathsf{v}) \to j$. This is a deterministic extraction algorithm that takes as input a trapdoor td, a function input $y$ and a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs an index $j \in [N]$.

$\mathsf{ExtractTag}(\mathsf{td}, y, \mathsf{v}) \to t$. This is a deterministic extraction algorithm that takes as input a trapdoor td, a function input $y$ and a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs a tag $t \in \{0,1\}^T$.

A PEHash is required to satisfy the following basic properties, and additional consistency properties.

**Definition 8.3** (PEHash Basic Properties)**.** *A $G$-functional predicate extractable hash family* PEHash *satisfies the following properties:*

**Completeness.** *For any $\lambda \in \mathbb{N}$, any $N, T, \ell \leq 2^\lambda$, any function $g \in G_\lambda$, any function input $y \in \{0,1\}^\ell$, any index $j \in [N]$, and any $x \in \{0,1\}^N$ and tags $\vec{t} = (t_1, \ldots, t_N) \in (\{0,1\}^T)^N$, let $f = g(y)$, then*

$$\Pr\left[\begin{array}{ll|ll} \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = f(x) \wedge & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g), \\ \mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, x_j, t_j, \rho) = 1 & : & \mathsf{v} = \mathsf{Hash}(\mathsf{hk}, y, x, \vec{t}), \\ & & \rho = \mathsf{Open}(\mathsf{hk}, y, x, \vec{t}, j) \end{array}\right] = 1.$$

**Succinctness.** *In the completeness experiment above, the size of the verification key* vk *and the hash value* v *is* $\mathsf{poly}(\lambda)$*. The size of the hash key* hk *is at most* $|g| \cdot \mathsf{poly}(\lambda)$*.*

**Computational binding.** *For any poly-size adversary $\mathcal{A}$ and polynomial $\ell(\lambda)$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$, any function $g \in G_\lambda$ and any function input $y \in \{0,1\}^\ell$,*

$$\Pr\left[\begin{array}{ll|ll} \mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, 0, t_0, \rho_0) = 1 \wedge & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g), \\ \mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, 1, t_1, \rho_1) = 1 & : & (\mathsf{v}, j, t_0, t_1, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

**Function hiding.** *For any poly-size adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$ and $g_0, g_1 \in G_\lambda$ such that $|g_0| = |g_1|$,*

$$\Pr\left[\begin{array}{ll|ll} \mathcal{A}(\mathsf{hk}, \mathsf{vk}) = b & : & b \leftarrow \{0,1\} \\ & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

**Definition 8.4** (PEHash Consistency Properties)**.** *The hash family is furthermore required to satisfy the following* consistency *properties:*

**Index extraction correctness.** *For any $\lambda \in \mathbb{N}$, any $\ell \leq 2^\lambda$, any function $g \in G_\lambda$ and function input $y \in \{0,1\}^\ell$ such that for $f = g(y) = (J, s)$ we have $J \neq \emptyset$, and any hash value* v*,*

$$\Pr\left[\begin{array}{ll|ll} \mathsf{ExtractIndex}(\mathsf{td}, y, \mathsf{v}) \in J & : & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g) \end{array}\right] = 1\,.$$

**Consistency of extraction.** *For any poly-size adversary $\mathcal{A}$ it holds that for any $g \in G_\lambda$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{ll|ll} \mathsf{Extract}(\mathsf{td}, y, \mathsf{v}) = 1 \ \wedge \ j \in J \wedge & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g), \\ \mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, 1 - s_j, t, \rho) = 1 & : & (y, \mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}), \\ & & (J, s) = g(y) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

*and*

$$\Pr\left[\begin{array}{ll|ll} \mathsf{Extract}(\mathsf{td}, y, \mathsf{v}) = 0 \wedge & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g), \\ \mathsf{ExtractIndex}(\mathsf{td}, y, \mathsf{v}) = j \wedge & : & (y, \mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}), \\ \mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, s_j, t, \rho) = 1 & & (J, s) = g(y) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

**Consistency of tag extraction.** *For any poly-size adversary $\mathcal{A}$ it holds that for any $g \in G_\lambda$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{ll|ll} \mathsf{Extract}(\mathsf{td}, y, \mathsf{v}) = 0 \wedge & & \\ \mathsf{ExtractIndex}(\mathsf{td}, y, \mathsf{v}) = j \wedge & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, g) \\ \mathsf{ExtractTag}(\mathsf{td}, y, \mathsf{v}) \neq t \wedge & : & (y, \mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \\ \mathsf{Verify}(\mathsf{vk}, y, \mathsf{v}, j, 1 - s_j, t, \rho) = 1 & & \end{array}\right]$$

**Theorem 8.5** ( [BBK+23], Theorem 5.3 and Remark 5.2)**.** *Assuming fully homomorphic encryption, there exists a* PEHash *family with tags with respect to the bit-fixing predicate family, which is $G$-functional with respect to any family $G$ of polynomial-size circuits.*

### 8.1.2 Functional Subset Extraction from Functional PEHash

We now recall the construction of SNARGs for monotone policy BatchNP from [BBK+23], and replace the PEHash with a functional PEHash to obtain a $F$-batch argument with $G$-functional subset extraction, for any family $F$ of monotone circuit policies and $G$ of polynomial size circuits. The construction uses the following building blocks:

- A $G'$-functional PEHash family with tags (Definition 8.4)

  $$(\mathsf{Gen}_{\mathsf{PEHT}}, \mathsf{Hash}_{\mathsf{PEHT}}, \mathsf{Open}_{\mathsf{PEHT}}, \mathsf{Verify}_{\mathsf{PEHT}}, \mathsf{Extract}_{\mathsf{PEHT}}, \mathsf{ExtractIndex}_{\mathsf{PEHT}}, \mathsf{ExtractTag}_{\mathsf{PEHT}})$$

  with respect to the bit-fixing predicate family, where $G'$ is a family of polynomial size circuits, containing a function $g'$ for every $g \in G_\lambda$, defined as follows:

  1. Receive an input $y \in \{0,1\}^\ell$ and compute $g(y) = J \subseteq [k]$.
  2. Output $f = (J, s)$ where $s = 0^J$ is the all-zero string.

- A $G''$-functional PEHash family

  $$(\mathsf{Gen}_{\mathsf{PEH}}, \mathsf{Hash}_{\mathsf{PEH}}, \mathsf{Open}_{\mathsf{PEH}}, \mathsf{Verify}_{\mathsf{PEH}}, \mathsf{Extract}_{\mathsf{PEH}}, \mathsf{ExtractIndex}_{\mathsf{PEH}})$$

  with respect to the bit-fixing predicate family, where $G''$ is a family of polynomial size circuits, containing a function $g''_i$ for every $g \in G_\lambda$ and $i \in [d]$, defined as follows:

  1. Receive an input $y' = (C, y)$ where $C \in F_\lambda$ is a monotone circuit, and $y \in \{0,1\}^\ell$ is an input to $g$.
  2. Compute $g(y) = J \subseteq [k]$.
  3. Let $(b^*_1, \ldots, b^*_N)$ be the values of all the wires in $C$ on input $\mathbb{1}_{[k]\setminus J}$.
  4. Let $J' \subseteq [N]$ be the set of wires in the $i$th layer of $C$ whose values $b^*$ are 0.
  5. Output $f = (J', s)$ where $s = 0^{J'}$ is the all-zero string.

- A seBARG scheme (Definition 3.5)

  $$(\mathsf{Gen}_{\mathsf{seBARG}}, \mathcal{P}_{\mathsf{seBARG}}, \mathcal{V}_{\mathsf{seBARG}}, \mathsf{Extract}_{\mathsf{seBARG}}).$$

We are now ready to describe our batch argument algorithms in Fig. 6.

The analysis of the construction above follows identically to the analysis of the somewhere-extractable SNARG for monotone policy BatchNP in [BBK+23] Theorem 7.1. In particular, the $G$-functional subset extraction property follows similarly to the somewhere argument of knowledge property in [BBK+23], and using the functional properties of the PEHash. We thus obtain Theorem 8.1.

$\mathsf{Gen}(1^\lambda, g)$ does the following:

1. Generate $(\mathsf{hk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{td}_{\mathsf{PEHT}}) \leftarrow \mathsf{Gen}_{\mathsf{PEHT}}(1^\lambda, g')$, for $g' \in G'_\lambda$ defined above.
2. For each $\beta \in \{1, 2\}$, generate $(\mathsf{hk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{td}^{(\beta)}_{\mathsf{PEH}}) \leftarrow \mathsf{Gen}_{\mathsf{PEH}}(1^\lambda, g''_\beta)$, where $g''_\beta$ is an arbitrary function in $G''_\lambda$.
3. Generate $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, I)$ where $I \subseteq [N]$ is initialized to $\{1, 2, N\}$.
4. Let $\mathsf{crs} = (\mathsf{hk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{hk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{hk}^{(2)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
5. Output $(\mathsf{crs}, \mathsf{td} = \mathsf{td}_{\mathsf{PEHT}})$.

$\mathcal{P}(\mathsf{crs}, y, C, M, z, 1^T, w_1, \ldots, w_k)$ does the following:

1. Let $(b_1, \ldots, b_k) \in \{0, 1\}^k$ such that $b_i = 1$ iff $M(z, i, w_i)$ accepts within $T$ steps.
2. Compute the values $(b_1, \ldots, b_N)$ of all the wires in the circuit $C$ on input $(b_1, \ldots, b_k)$.
3. Parse $\mathsf{crs} = (\mathsf{hk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{hk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{hk}^{(2)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
4. Compute $\mathsf{v}_{\mathsf{PEHT}} = \mathsf{Hash}_{\mathsf{PEHT}}(\mathsf{hk}_{\mathsf{PEHT}}, y, (b_1, \ldots, b_k), (w_1, \ldots, w_k))$.
5. Compute $\mathsf{v}^{(\beta)} = \mathsf{Hash}_{\mathsf{PEH}}(\mathsf{hk}^{(\beta)}_{\mathsf{PEH}}, (C, y), (b_1, \ldots, b_N))$ for $\beta \in \{1, 2\}$.
6. Define an instance $X = (M', z', N, T)$ of $\mathsf{BatchIndexTMSAT}$. The input $z'$ is defined as $z' = (C, x_1, \ldots, x_k, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta \in \{1,2\}})$. The batch size is set to $N$. The Turing machine $M'(z', j, w'_j)$ is defined to operate as follows:
   (a) Parse $z' = (C, M, z, T, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta \in \{1,2\}})$.
   (b) If $1 \le j \le k$:
      i. Parse $w'_j = (w_j, b_j, \rho_{\mathsf{PEHT}}, \rho^{(1)}, \rho^{(2)})$.
      ii. Check that $\mathsf{Verify}_{\mathsf{PEHT}}(\mathsf{vk}_{\mathsf{PEHT}}, y, \mathsf{v}_{\mathsf{PEHT}}, j, b_j, w_j, \rho_{\mathsf{PEHT}}) = 1$.
      iii. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, (C, y), \mathsf{v}^{(\beta)}, j, b_j, \rho^{(\beta)}) = 1$ for $\beta \in \{1, 2\}$.
      iv. Check that $b_j = 1$ iff $M(z, j, w_j)$ accepts within $T$ steps.
   (c) If $j > k$:
      i. Compute the $j$th gate of $C$, $g_j = (j, j_1, j_2, c \in \{\mathrm{AND}, \mathrm{OR}\})$.
      ii. Parse $w'_j = \left(b_j, b_{j_1}, b_{j_2}, \left(\rho^{(\beta)}_j, \rho^{(\beta)}_{j_1}, \rho^{(\beta)}_{j_2}\right)_{\beta \in \{1,2\}}\right)$.
      iii. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, (C, y), \mathsf{v}^{(\beta)}, j, b_j, \rho^{(\beta)}_j) = 1$ for $\beta \in \{1, 2\}$.
      iv. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, (C, y), \mathsf{v}^{(\beta)}, j_\alpha, b_{j_\alpha}, \rho^{(\beta)}_{j_\alpha}) = 1$ for $\alpha, \beta \in \{1, 2\}$.
      v. Check that $b_j = c(b_{j_1}, b_{j_2})$. (That is, check that the gate is satisfied.)
      vi. If $j = N$ is the output wire then check that $b_j = 1$.

   The description length of $M$ is a constant. Finally, the time bound $T'$ is set so that the pseudocode above terminates.
7. For every $j \in [N]$, construct a witness $(j, w'_j)$ for $X$, using the $\mathsf{Open}_{\mathsf{PEH}}$, $\mathsf{Open}_{\mathsf{PEHT}}$ algorithms to produce openings for $(b_1, \ldots, b_N)$ and $(b_1, \ldots, b_k)$ with tags $(w_1, \ldots, w_k)$ as appropriate.
8. Compute $\pi_{\mathsf{seBARG}} = \mathcal{P}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, M', z', 1^{T'}, w'_1, \ldots, w'_N)$.
9. Output $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.

$\mathcal{V}(\mathsf{crs}, y, x, \pi)$ does the following:

1. Parse $\mathsf{crs}_{\mathcal{V}} = (\mathsf{vk}_{\mathsf{PEHT}}, \mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
2. Parse $x = (C, M, z, T)$.
3. Parse $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
4. Define $X = (M', z', N, T')$ as above.
5. Output $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, X, \pi_{\mathsf{seBARG}})$.

$\mathsf{Extract}(\mathsf{td}, y, \pi)$ does the following:

1. Parse $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
2. Compute $j = \mathsf{ExtractIndex}_{\mathsf{PEHT}}(\mathsf{td}, y, \mathsf{v}_{\mathsf{PEHT}})$, and $w_j = \mathsf{ExtractTag}(\mathsf{td}, y, \mathsf{v}_{\mathsf{PEHT}})$.
3. Output $(j, w_j)$.

Figure 6: Construction of BARGs with Functional Subset Extraction

# References


[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 252–283. Springer, Heidelberg, August 2023. 2, 4, 5, 6, 8, 9, 10, 15, 18, 22, 23, 25, 53, 54, 56, 63

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017. 2

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013. 2

[BCPR16] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. *SIAM J. Comput.*, 45(5):1910–1952, 2016. 2

[BDKP22] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable private information retrieval. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part III*, volume 13749 of *LNCS*, pages 3–32. Springer, Heidelberg, November 2022. 11, 13, 35, 37, 39, 42, 68, 69

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003. 1, 5

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. 5

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 474–482. ACM Press, June 2017. 2, 7

[BKP+24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. 2024. 21

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001. 5

[BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. *IACR Cryptol. ePrint Arch.*, page 833, 2022. 4

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Heidelberg, August 1988. 5

[CDM00] Ronald Cramer, Ivan Damgård, and Philip D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. *IACR Cryptol. ePrint Arch.*, page 45, 2000. 6

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 635–668. Springer, Heidelberg, August 2023. 2, 6, 19

[Chi] Chia Network. BLS signatures in C++ using the RELIC toolkit. https://github.com/Chia-Network/bls-signatures. Accessed: 2019-05-06. 1

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg. 2, 6

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022. 2, 6, 7, 8, 16, 18, 19, 22

[CMRR23] Lily Chen, Dustin Moody, Andrew Regenscheid, and Angela Robinson. Digital signature standard (dss). 2023. https://www.nist.gov/publications/digital-signature-standard-dss-3. 5

[Cra97] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD Thesis, University of Amsterdam, January 1997. 6

[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. *IACR Cryptol. ePrint Arch.*, page 598, 2023. 5

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, August 1988. 1, 5

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990. 1, 5

[DFI] DFINITY. go-dfinity-crypto. https://github.com/dfinity/go-dfinity-crypto. Accessed: 2019-05-06. 1

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022. 2, 4, 6

[dra17] drand: Randomness Beacon Service, 2017. https://drand.love/docs/cryptography/. 1

[EJN17] Steve Ellis, Ari Juels, and Sergey Nazarov. Chainlink: A decentralized oracle network. *Retrieved March*, 11:2018, 2017. 1

[FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 513–530. Springer, Heidelberg, August 2013. 5

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. 6

[GJM+23] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. *IACR Cryptol. ePrint Arch.*, page 567, 2023. 5

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. 5

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 520–549. Springer, Heidelberg, May / June 2022. 6

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 3–34. Springer, Heidelberg, April 2015. 4, 5

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. 54

[IN83] K Itakura and K Nakamura. A public key cryptosystem suitable for digital multisignatures. In *NEC Research and Development*, 1983. 1, 5

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1545–1552. ACM, 2023. 2, 6, 18, 19

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. 6

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 652–673. Springer, Heidelberg, August 2020. 2

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Heidelberg, November 2021. 18

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Heidelberg, May 2004. 5

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Heidelberg, May / June 2006. 5

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988. 17

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 2

[NHN+19] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*, 7:85727–85745, 2019. 1

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 33–43. ACM, 1989. 16

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Heidelberg, November / December 2015. 54

[Pol20] Poly Network. Poly Network, 2020. https://poly.network/. 1

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd FOCS*, pages 1045–1056. IEEE Computer Society Press, October / November 2022. 6

[RS09] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *ISA*, volume 5576 of *Lecture Notes in Computer Science*, pages 750–759. Springer, 2009. 5

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990. 5

[sep21] Threshold Signature Wallets, 2021. https://sepior.com/mpc-blog/threshold-signature-wallets. 1

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979. 5

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000. 4

[Tou84] Sam Toueg. Randomized byzantine agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, page 163–178. Association for Computing Machinery, 1984. 1

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Heidelberg, August 2022. 2, 6, 19

# A BARGs with Adaptive Subset Extraction for Low-Depth Policies

In this section we give a batch argument with adaptive subset extraction for families of monotone policies implemented by a low-depth monotone circuits. The construction is identical to the SNARGs for low-depth monotone BatchNP circuits in [BBK+23]. We show that with minor modifications to the proof of soundness, their scheme satisfies adaptive subset extraction (as in Definition 5.2). Most of the construction and proof are taken verbatim from [BBK+23].

We proceed by defining low-depth monotone circuit families, then state our main theorem and a corollary.

**Definition A.1** (Depth-$d$ Monotone Circuit Policies)**.** *Let $d(\lambda)$ be a function and let $F = \{F_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of functions. $F$ is a depth-$d$ family of policies if there exist polynomials $N(\lambda), k(\lambda)$ such that each $f \in F_\lambda$ is a monotone function $f\colon \{0,1\}^k \to \{0,1\}$ computable by a monotone circuit $C$ of size $N$ and depth $d$.*

**Theorem A.2.** *Assuming a somewhere extractable batch argument scheme and a hash family with local opening, there exists a scheme* $\mathsf{BARG} = (\mathsf{Setup}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *such that for any family $F$ of depth-$d$ policies where $d = O(\log\lambda)$,* BARG *is a $F$-batch argument with $O(k \cdot 2^d)$-adaptive subset extraction for relation $\mathcal{R}_{\mathsf{full}}$.*

**Corollary A.3.** *Assuming a somewhere extractable batch argument scheme and a hash family with local opening, there exists a scheme* $\mathsf{AggS} = (\mathsf{Setup}, \mathsf{KeyAgg}, \mathsf{SigAgg}, \mathsf{AggVerify})$ *such that for any family $F$ of depth-$d$ policies where $d = O(\log\lambda)$ and for any digital signature scheme* S, AggS *is a $F$-aggregation scheme for* S.

In the rest of this section, we prove Theorem A.2. Using Theorem 5.4 we immediately obtain Corollary A.3.

In what follows we construct $F$-batch arguments with adaptive subset extraction for low-depth monotone circuits for relation $\mathcal{R}_{\mathsf{full}}$. Our construction uses the following building blocks:

- A hash family with local opening (Definition 3.2)

$$(\mathsf{Gen}_{\mathsf{HT}}, \mathsf{Hash}_{\mathsf{HT}}, \mathsf{Open}_{\mathsf{HT}}, \mathsf{Verify}_{\mathsf{HT}}) .$$

- A somewhere extractable batch argument scheme (Definition 3.5)

$$(\mathsf{Gen}_{\mathsf{seBARG}}, \mathcal{P}_{\mathsf{seBARG}}, \mathcal{V}_{\mathsf{seBARG}}, \mathsf{Extract}_{\mathsf{seBARG}}) .$$

We now describe the batch argument algorithms.

$\mathsf{Setup}(1^\lambda)$ does the following:

1. Generate $\mathsf{hk}_{\mathsf{HT}} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^\lambda)$.
2. For every $t \in [\lambda]$, sample $i_t \leftarrow [2^t]$ uniformly at random.
3. For every $t \in [\lambda]$, let $(\mathsf{crs}_{\mathsf{seBARG},t}, \mathsf{td}_{\mathsf{seBARG},t}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, I)$ where $I = \{i_t, N\}$.
4. Output $\mathsf{crs} = (\mathsf{hk}_{\mathsf{HT}}, (\mathsf{crs}_{\mathsf{seBARG},t})_{t\in[\lambda]})$, $\mathsf{td} = (i_t, \mathsf{td}_{\mathsf{seBARG},t})_{t\in[\lambda]}$.

$\mathcal{P}(\mathsf{crs}, C, M, z, 1^T, w_1, \ldots, w_k)$ does the following:

1. Parse $\mathsf{crs} = (\mathsf{hk}_{\mathsf{HT}}, (\mathsf{crs}_{\mathsf{seBARG},t})_{t\in[\lambda]})$.

2. Let $\mathsf{crs}_{\mathsf{seBARG}} = \mathsf{crs}_{\mathsf{seBARG},t}$ for $t = \lceil \log k \rceil$.
3. Compute the values of all the wires in the circuit $C$ on the input $(b_1, \ldots, b_k)$, where $b_i = M(z, i, w_i)$. Denote these values by $(b_1, \ldots, b_N)$.
4. Compute $\mathsf{rt} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, (b_1, \ldots, b_N))$.
5. Define an instance $X = (M', z', N, T')$ of BatchIndexTMSAT. The input $z'$ is defined as $z' = (C, M, z, T, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$. The batch size is set to $N$, the size of the circuit $C$. The Turing machine $M'(z', j, w'_j)$ is defined to operate as follows:
   (a) Parse $z' = (C, M, z, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$.
   (b) If $1 \leq j \leq k$:
      i. Parse $w'_j = (w_j, b_j, \rho)$.
      ii. Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j, b_j, \rho) = 1$.
      iii. Check that $b_j = 1$ iff $M(z, j, w_j)$ accepts within $T$ steps.
   (c) If $j > k$:
      i. Compute the $j$th gate of $C$, $g_j = \big(j, j_1, j_2, c_j \in \{\mathrm{AND}, \mathrm{OR}\}\big)$.
      ii. Parse $w'_j = \big(b_j, b_{j_1}, b_{j_2}, \rho_j, \rho_{j_1}, \rho_{j_2}\big)$.
      iii. Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j, b_j, \rho_j) = 1$.
      iv. Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j_\alpha, b_{j_\alpha}, \rho_j) = 1$ for $\alpha \in \{1, 2\}$.
      v. Check that $b_j = c_j(b_{j_1}, b_{j_2})$. (That is, check that the gate is satisfied.)
      vi. If $j = N$ is the output wire then check that $b_j = 1$.

   The description length of $M$ is a constant. Finally, the time bound $T'$ is set so that the pseudocode above terminates.
6. For every $j \in [N]$, construct a witness $w'_j$ for $X$, using the $\mathsf{Open}_{\mathsf{HT}}$ algorithm to produce openings for $(b_1, \ldots, b_N)$ as appropriate.
7. Compute $\pi_{\mathsf{seBARG}} = \mathcal{P}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, M', z', 1^{T'}, w'_1, \ldots, w'_N)$.
8. Output $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.

$\mathcal{V}(\mathsf{crs}, x, \pi)$ does the following:

1. Parse $\mathsf{crs} = (\mathsf{hk}_{\mathsf{HT}}, (\mathsf{crs}_{\mathsf{seBARG},t})_{t \in [\lambda]})$, $x = (C, M, z, T)$ and $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
2. Let $\mathsf{crs}_{\mathsf{seBARG}} = \mathsf{crs}_{\mathsf{seBARG},t}$ for $t = \lceil \log k \rceil$.
3. Define $X = (M', z', N, T')$ as above.
4. Output $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, X, \pi_{\mathsf{seBARG}})$.

$\mathsf{Extract}(\mathsf{td}, \pi)$ does the following:

1. Parse $\mathsf{td} = (i_t, \mathsf{td}_{\mathsf{seBARG},t})_{t \in [\lambda]}$ and $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
2. Let $(i, \mathsf{td}_{\mathsf{seBARG}}) = (i_t, \mathsf{td}_{\mathsf{seBARG},t})$ for $t = \lceil \log k \rceil$.
3. Extract $w'_i, w'_N = \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$.
4. Parse $w'_i = (w_i, b_i, \rho)$.
5. Output $(i, w_i)$.

**Proof of Theorem A.2.** Let $d = O(\log \Lambda)$ and let $F$ be a family of depth-$d$ monotone circuit policies. We prove that the construction above is a $F$-batch argument with $O(k \cdot 2^d)$-adaptive subset extraction for relation $\mathcal{R}_{\mathsf{full}}$.

**Completeness.** Follows directly from the completeness of the underlying hash family with local opening and seBARG.

**Succinctness.** By the seBARG and hash family efficiency we have $|\pi| \leq m \cdot \mathsf{poly}(\lambda)$.

**$O(k \cdot 2^d)$-Adaptive Subset Extraction.** Fix any polynomial $T(\lambda)$, any poly-size cheating prover $\mathcal{P}^*$, and any sequence $\{f_\lambda \in F_\lambda\}_{\lambda \in \mathbb{N}}$ computable by circuits $\{C_\lambda\}_{\lambda \in \mathbb{N}}$. Let $\mathsf{EXP}$ be the experiment defined in the adaptive subset extraction requirement:

- Generate $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- Run the cheating prover and obtain $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
- Let $x = (f_\lambda, M, z, T)$.
- Extract $(i, w_i) \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)$.

For $1 \leq j_1, j_2 \leq N$, we define an alternative experiment $\mathsf{EXP}_{j_1,j_2}$ as follows:

- Sample $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}_{j_1,j_2}(1^\lambda)$, where $\mathsf{Setup}_{j_1,j_2}$ is identical to Setup, except that for $t = \lceil \log k \rceil$ it runs $\mathsf{Gen}_{\mathsf{seBARG}}$ on $\{j_1, j_2\}$ rather than on $\{i, N\}$ for a random $i$.
- Run the prover and obtain $(M, z, \pi, J) \leftarrow \mathcal{P}^*(\mathsf{crs})$.
- Let $x = (f_\lambda, M, z, T)$.
- Extract $w'^{(1)}_{j_1}, w'^{(2)}_{j_2} \leftarrow \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$, where we parse $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
- For $\alpha \in \{1, 2\}$, parse:
  1. If $1 \leq j \leq k$, parse $w'^{(\alpha)}_{j_\alpha} = (w^{(\alpha)}_{j_\alpha}, b^{(\alpha)}_{j_\alpha}, \rho^{(\alpha)}_{j_\alpha})$.
  2. If $j > k$, parse $w'^{(\alpha)}_{j_\alpha} = (b^{(\alpha)}_{j_\alpha}, b^{(\alpha)}_{j_{\alpha,1}}, b^{(\alpha)}_{j_{\alpha,2}}, \rho^{(\alpha)}_{j_\alpha}, \rho^{(\alpha)}_{j_{\alpha,1}}, \rho^{(\alpha)}_{j_{\alpha,2}})$.

For ease of notation, if $j_1 = j_2 = j$, we use the notation $\mathsf{EXP}_j$ instead of $\mathsf{EXP}_{j,j}$, and skip the superscript (1) when referring to $w'^{(1)}_j$ and the values parsed from it.

For a wire $j$ in the circuit $C$ computing function $f$, we denote by $C_j(x) \in \{0, 1\}$, the value of the $j$th wire in $C$ when computing the circuit on the input $x$.

**Lemma A.4.** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, there exists an input wire* $j \in [k]$ *such that*

$$\Pr_{\mathsf{EXP}_j}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ b_j > C_j(\mathbb{1}_{[k]\setminus J})\end{array}\right] \geq \frac{1}{2^{d(\lambda)}} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mathsf{negl}(\lambda)\,.$$

Before proving Lemma A.4, we first argue that the lemma indeed implies $O(k \cdot 2^d)$-adaptive subset extraction. Since $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ implies that $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, X, \pi_{\mathsf{seBARG}}) = 1$, by the seBARG somewhere argument of knowledge we get that except with negligible probability, $M'(z', j, w'_j) = 1$, which means that $M(z, j, w_j) = b_j = 1$. So, we get

$$\Pr_{\mathsf{EXP}_j}\left[\ j \in J \ \wedge\ M(z, j, w_j) = 1\ \right] \geq \frac{1}{2^{d(\lambda)}} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mathsf{negl}(\lambda)\,.$$

By seBARG index hiding, the same holds when switching to the experiment $\mathsf{EXP}_{j,N}$, except with negligible probability. We recall that $\mathsf{EXP}$ is identical to running $\mathsf{EXP}_{i,N}$ for a random $i \in [2^t]$ for $t = \lceil \log k \rceil$, and in particular with probability $\frac{1}{2^t} \geq \frac{1}{2k}$ we get the experiment $\mathsf{EXP}_{j,N}$, so

$$\Pr_{\mathsf{EXP}}\left[\ i \in J \ \wedge\ M(z, i, w_i) = 1\ \right] > \frac{1}{2k(\lambda)}\left(\frac{1}{2^{d(\lambda)}} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mathsf{negl}(\lambda)\right)\,,$$

which proves $O(k \cdot 2^d)$-adaptive subset extraction.

**Proof of Lemma A.4.** Let $\mu_1, \mu_2$ be the negligible functions in the seBARG index hiding and somewhere argument of knowledge properties (for the adversary $\mathcal{P}^*$ and corresponding polynomials $k, n, m, T$), and $\mu_3$ be the negligible function in the hash tree collision resistance wrt opening property. Define $\mu_i(\lambda) = 2(d(\lambda) - i + 1)(\mu_1(\lambda) + \mu_2(\lambda) + \mu_3(\lambda))$.

It suffices to prove the following inductive claim: for every $\lambda \in \mathbb{N}$, let $C$ be the circuit computing $f_\lambda$ then for every $0 \leq i \leq d(\lambda)$, there exists a wire $j$ in the $i$th layer of $C$ such that

$$\Pr_{\mathsf{EXP}_j}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ b_j > C_j(\mathbb{1}_{[k]\setminus J})\end{array}\right] \geq \frac{1}{2^{d(\lambda)-i}} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mu_i(\lambda)\,.$$

First, for $i = d$, we take the output wire $j = N$. By the index hiding of the seBARG we have that

$$\Pr_{\mathsf{EXP}_N}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] \geq \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mu_1(\lambda)\,.$$

Now, by the seBARG argument of knowledge, we have that $\mathcal{V}(\mathsf{crs}, x, \pi)$ implies $M'(z', N, w'_N) = 1$ except with negligible probability $\mu_2$, which implies that $b_N = 1$. So, since $C_N = f$, we get

$$\Pr_{\mathsf{EXP}_N}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ b_N > C_N(\mathbb{1}_{[k]\setminus J})\end{array}\right] \geq \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mu_1(\lambda) - \mu_2(\lambda)\,.$$

This proves the base case. Now assume by induction that the lemma holds for $i+1$, and let $j$ in the $i+1$th layer of $C$ given by the lemma. By the somewhere argument of knowledge property of the underlying seBARG, we have

$$\Pr_{\mathsf{EXP}_j}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ M'(z', j, w'_j) = 1 \wedge \\ b_j > C_j(\mathbb{1}_{[k]\setminus J})\end{array}\right] \geq \frac{1}{2^{d(\lambda)-i-1}} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mu_{i+1}(\lambda) - \mu_2(\lambda)\,.$$

We now analyze the $j$th gate. We know that $M'(z', j, w'_j) = 1$ implies $b_j = c_j(b_{j_1}, b_{j_2})$. Moreover, by monotonicity, if we also have $b_j > C_j(\mathbb{1}_{[k]\setminus J})$, we get $b_{j_\alpha} > C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})$ for some $\alpha \in \{1, 2\}$. Therefore, there exists an $\alpha \in \{1, 2\}$ for which we have

$$\Pr_{\mathsf{EXP}_j}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ b_{j_\alpha} > C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\end{array}\right] \geq \frac{1}{2^{d(\lambda)-i}} \cdot \Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge \\ f(\mathbb{1}_{[k]\setminus J}) = 0\end{array}\right] - \mu_{i+1}(\lambda) - \mu_2(\lambda)\,. \tag{4}$$

It remains to prove the following claim, which shows that the lemma holds for $j_\alpha$, and thus finishes the inductive step.

**Claim A.5.**

$$\Pr_{\mathsf{EXP}_{j_\alpha}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},\mathbf{x},\pi)=1\ \wedge\\ b_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\end{array}\right]\geq\frac{1}{2^{d(\lambda)-i}}\cdot\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ f(\mathbb{1}_{[k]\setminus J})=0\end{array}\right]-\mu_i(\lambda)\,.$$

**Proof of Claim A.5.** Recall that $\mathsf{EXP}_j$ is a shorthand for $\mathsf{EXP}_{j,j}$, and rewrite Eq. (4) as follows:

$$\begin{aligned}&\Pr_{\mathsf{EXP}_{j,j}}\left[\ \mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\ b^{(1)}_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\ \right]\\ &\geq\frac{1}{2^{d(\lambda)-i}}\cdot\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ f(\mathbb{1}_{[k]\setminus J})=0\end{array}\right]-\mu_{i+1}(\lambda)-\mu_2(\lambda)\,.\end{aligned}$$

We first switch to the hybrid experiment $\mathsf{EXP}_{j,j_\alpha}$. By the index hiding of seBARG, we have

$$\begin{aligned}&\Pr_{\mathsf{EXP}_{j,j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\ b^{(1)}_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\ \right]\\ &\geq\frac{1}{2^{d(\lambda)-i}}\cdot\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ f(\mathbb{1}_{[k]\setminus J})=0\end{array}\right]-\mu_{i+1}(\lambda)-\mu_2(\lambda)-\mu_1(\lambda)\,.\end{aligned}$$

Let $\mathsf{Correct}_M$ denote the event that $M'(z',j,w'^{(1)}_j)=1$ and $M'(z',j_\alpha,w'^{(2)}_{j_\alpha})=1$. By the somewhere argument of knowledge property of the seBARG, we have that in the above experiment, $\mathsf{Correct}_M$ holds except with negligible probability $\mu_2$.

Now, observe that $M(z,j,w'^{(1)}_j)=1$ implies that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}},\mathsf{rt},j_\alpha,b^{(1)}_{j_\alpha},\rho^{(1)}_{j_\alpha})=1$, whereas $M(z,j_\alpha,w'^{(2)}_{j_\alpha})=1$ implies $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}},\mathsf{rt},j_\alpha,b^{(2)}_{j_\alpha},\rho^{(2)}_{j_\alpha})=1$. By the collision resistance wrt opening property of the HT family, we get that except with negligible probability, $b^{(1)}_{j_\alpha}=b^{(2)}_{j_\alpha}$. Therefore,

$$\begin{aligned}&\Pr_{\mathsf{EXP}_{j,j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\ b^{(2)}_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\ \right]\\ &\geq\frac{1}{2^{d(\lambda)-i}}\cdot\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ f(\mathbb{1}_{[k]\setminus J})=0\end{array}\right]-\mu_{i+1}(\lambda)-\mu_2(\lambda)-\mu_1(\lambda)-\mu_3(\lambda)\,.\end{aligned}$$

Finally, we switch to the experiment $\mathsf{EXP}_{j_\alpha,j_\alpha}$. By the index hiding of seBARG,

$$\begin{aligned}&\Pr_{\mathsf{EXP}_{j_\alpha,j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\ b^{(2)}_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\ \right]\\ &\geq\frac{1}{2^{d(\lambda)-i}}\cdot\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ f(\mathbb{1}_{[k]\setminus J})=0\end{array}\right]-\mu_{i+1}(\lambda)-\mu_2(\lambda)-2\mu_1(\lambda)-\mu_3(\lambda)\,.\end{aligned}$$

Applying the same argument using somewhere argument of knowledge and collision resistance wrt opening, we get that in the above experiment we also have $b^{(1)}_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})$ except with negligible probability, which implies

$$\Pr_{\mathsf{EXP}_{j_\alpha}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ b_{j_\alpha}>C_{j_\alpha}(\mathbb{1}_{[k]\setminus J})\end{array}\right]\geq\frac{1}{2^{d(\lambda)-i}}\cdot\Pr_{\mathsf{EXP}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs},x,\pi)=1\ \wedge\\ f(\mathbb{1}_{[k]\setminus J})=0\end{array}\right]-\mu_i(\lambda)\,.$$

This concludes the proof of Claim A.5, Lemma A.4 and Theorem A.2.

# B Verifiable Private Information Retrieval (vPIR)

In this section we define vPIR [BDKP22]. For simplicity and consistency with Section 6, we make the following modifications:

- The Setup and Query algorithms are merged.
- We do not consider verification with multiple queries.
- We only consider publicly verifiable vPIR schemes.
- We only consider read-once bounded-space constraints.
- We consider simulation security for non-adaptive constraints. This differs from the definition in [BDKP22], that considers adaptively chosen constraints, which requires limiting the constraint's description length to $N = O(\log \Lambda)$. However, as noted in a remark in their work (and as we prove in Section 6 where we analyze a stronger form of vPIR), simulation security for non-adaptive constraints holds even when their length is $N \leq \Lambda$.

**Read-once bounded-space constraints.** For parameters $T, S$, we define a read-once bounded-space constraint represented as a Turing machine $\Gamma \in \{0,1\}^N$. We say that a database $D = (r_i)_{i \in [k]} \in \{0,1\}^{k \times \omega}$ satisfies $\Gamma$ if and only if for every $i \in [k-1]$ there exists $c_i \in \{0,1\}^S$ such that $\Gamma(c_{i-1}, r_i)$ outputs $c_i$ within $T$ steps where $c_0, c_k$ are some fixed starting and accepting configurations. We denote by $U_{T,S}(\Gamma, D)$ the bit that indicates whether or not $D$ satisfies the constraint $\Gamma$.

**Syntax.** A verifiable private information retrieval (vPIR) scheme consists of following polynomial-time algorithms:

$\mathsf{Query}(1^\lambda, t) \to (\mathsf{dk}, \mathsf{vk}, q)$**.** This is a probabilistic algorithm that takes as input the security parameter $1^\lambda$ and a row index $t \in [2^\lambda]$. It outputs a decryption key $\mathsf{dk}$, a verification key $\mathsf{vk}$ and a query $q$.

$\mathsf{Answer}(D, 1^T, \Gamma, q) \to a$**.** This is a deterministic algorithm that takes as input a database $D \in \{0,1\}^{k \times \omega}$, a time bound $1^T$, a constraint $\Gamma$, and a query $q$. It outputs an answer $a$.

$\mathsf{Dec}(\mathsf{dk}, a) \to r$**.** This is a deterministic algorithm that takes as input a decryption key $\mathsf{dk}$, and an answer $a$. It outputs a row $r \in \{0,1\}^\omega$.

$\mathsf{Verify}(\mathsf{vk}, \Gamma, a) \to 0/1$**.** This is a deterministic algorithm that takes as input a verification key $\mathsf{vk}$, a constraint $\Gamma$, and an answer $a$. It outputs a bit (1 to accept, 0 to reject).

**Definition B.1.** *A $\Lambda$-secure* vPIR *scheme* (Query, Answer, Dec, Verify) *for $\mathcal{U}$ is required to satisfy the following properties:*

**Completeness.** *For any $\lambda \in \mathbb{N}$, any $T, N, S, k, \omega \leq 2^\lambda$, database $D \in \{0,1\}^{k \times w}$, row index $t \in [k]$ and constraint $\Gamma \in \{0,1\}^N$ such that $U_{T,S}(\Gamma, D) = 1$,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Dec}(\mathsf{dk}, a) = D[t] \wedge & (\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t) \\ \mathsf{Verify}(\mathsf{vk}, \Gamma, a) = 1 & a \leftarrow \mathsf{Answer}(D, 1^T, \Gamma, q) \end{array}\right] = 1 \ .$$

**Efficiency.** *In the completeness experiment above,* $|\mathsf{vk}|+|q|+|a| \leq \omega \cdot \mathrm{poly}(\lambda, \log(kT))$.

**$\Lambda$-Privacy.** *For any* $\mathrm{poly}(\Lambda)$*-size adversary* $\mathcal{A}$ *and function* $k(\lambda) \leq \Lambda(\lambda)$ *there exists a negligible function* $\mathsf{negl}$ *such that for any* $\lambda \in \mathbb{N}$ *and row indices* $t_0, t_1 \in [k(\lambda)]$,

$$\Pr\left[\ \mathcal{A}(\mathsf{vk}, q) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\} \\ (\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t_b) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\Lambda)\ .$$

**$\Lambda$-Simulation security.** *For any functions* $T(\lambda), N(\lambda), k(\lambda), \omega(\lambda) \leq \Lambda(\lambda)$*, function* $S(\lambda) = O(\log \Lambda)$*, any* $\mathrm{poly}(\Lambda)$*-size adversary* $\mathcal{A}$ *and polynomial* $P$ *there exists a* $\mathrm{poly}(\Lambda)$*-size simulator* $\mathsf{Sim}$ *such that for any* $\mathrm{poly}(\Lambda)$*-size distinguisher* $\mathcal{D}$*,* $\lambda \in \mathbb{N}$*, constraint* $\Gamma$ *and row index* $t \in [k]$,

$$\left|\Pr\left[\ \mathcal{D}(\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma, t)) = 1\ \right] - \Pr\left[\ \mathcal{D}(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma, t)) = 1\ \right]\right| < \frac{1}{P(\Lambda)}\ ,$$

*where the experiments* $\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma, t)$ *and* $\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma, t)$ *are defined as follows:*

$\mathsf{Real}_{\mathcal{A}}(\lambda, \Gamma, t)$**:**

- *Generate a query* $(\mathsf{dk}, \mathsf{vk}, q) \leftarrow \mathsf{Query}(1^\lambda, t)$.
- *Run the adversary and obtain* $a \leftarrow \mathcal{A}(\mathsf{vk}, q)$.
- *If* $\mathsf{Verify}(\mathsf{vk}, \Gamma, a) = 1$ *output* $\mathsf{Dec}(\mathsf{dk}, a)$*. Otherwise output* $\perp$.

$\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \Gamma, t)$**:**

- *Run the simulator and obtain* $D \leftarrow \mathsf{Sim}(\lambda, \Gamma)$.
- *If* $U_{T,S}(\Gamma, D) = 1$ *output* $D[t]$*. Otherwise output* $\perp$.

**Theorem B.2** ( [BDKP22])**.** *Assuming a* $\Lambda$*-secure somewhere extractable batch argument scheme and a* $\Lambda$*-secure hash family with local opening, there exists a* $\Lambda$*-secure* vPIR *scheme for read-once bounded-space constraints.*