

# Rate-1 Non-Interactive Arguments for Batch-NP and Applications \*

Lalita Devadas  
MIT

Rishab Goyal  
University of Wisconsin-Madison

Yael Kalai  
Microsoft Research and MIT

Vinod Vaikuntanathan  
MIT

## Abstract

We present a *rate-1* construction of a publicly verifiable non-interactive argument system for batch-NP (also called a BARG), under the LWE assumption. Namely, a proof corresponding to a batch of  $k$  NP statements each with an  $m$ -bit witness, has size  $m + \text{poly}(\lambda, \log k)$ .

In contrast, prior work either relied on non-standard knowledge assumptions, or produced proofs of size  $m \cdot \text{poly}(\lambda, \log k)$  (Choudhuri, Jain, and Jin, STOC 2021, following Kalai, Paneth, and Yang 2019).

We show how to use our rate-1 BARG scheme to obtain the following results, all under the LWE assumption in the standard model:

- A multi-hop BARG scheme for NP.
- A multi-hop aggregate signature scheme.
- An incrementally verifiable computation (IVC) scheme for arbitrary  $T$ -time deterministic computations with proof size  $\text{poly}(\lambda, \log T)$ .

Prior to this work, multi-hop BARGs were only known under non-standard knowledge assumptions or in the random oracle model; aggregate signatures were only known under indistinguishability obfuscation (and RSA) or in the random oracle model; IVC schemes with proofs of size  $\text{poly}(\lambda, T^\epsilon)$  were known under a bilinear map assumption, and with proofs of size  $\text{poly}(\lambda, \log T)$  were only known under non-standard knowledge assumptions or in the random oracle model.

---

\*All authors were supported in part by DARPA under Agreement No. HR00112020023, and Devadas, Goyal, and Vaikuntanathan were supported in part by a grant from the MIT-IBM Watson AI, a grant from Analog Devices, a Microsoft Trustworthy AI grant, and a Thornton Family Faculty Research Innovation Fellowship from MIT. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. The work of Goyal was done when he was at MIT.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Our Main Technical Result: Rate-1 seBARG for NP . . . . .	2
1.2	Our Main Tool: Fully-Local Somewhere Extractable Hash (SEH) Families . . . . .	3
1.3	Applications of seBARGs . . . . .	4
<b>2</b>	<b>Technical Overview</b>	<b>9</b>
2.1	Main Ingredient: Flexible RAM SNARGs with Partial Input Soundness . . . . .	9
2.2	Our Rate-1 seBARG Scheme . . . . .	10
2.3	Our Fully-Local SEH Family . . . . .	11
<b>3</b>	<b>Preliminaries</b>	<b>13</b>
3.1	Somewhere Extractable Hash (SEH) Families . . . . .	13
3.2	Somewhere Extractable Batch Arguments (seBARGs) . . . . .	15
3.3	Flexible RAM SNARGs with Partial Input Soundness . . . . .	17
3.4	Homomorphic Encryption with Ciphertext Compression . . . . .	18
<b>4</b>	<b>Rate-1 Fully-Local SEH (flSEH) Families</b>	<b>21</b>
4.1	Definition . . . . .	21
4.2	Construction . . . . .	23
4.3	Analysis . . . . .	29
<b>5</b>	<b>Rate-1 seBARGs</b>	<b>34</b>
<b>6</b>	<b>Multi-Hop seBARGs</b>	<b>37</b>
6.1	Definition . . . . .	38
6.2	Construction and Analysis . . . . .	40
6.3	Hashed Multi-Hop seBARGs . . . . .	44
6.3.1	Definition . . . . .	44
6.3.2	Construction and Analysis . . . . .	46
<b>7</b>	<b>Applications</b>	<b>50</b>
7.1	Aggregate Signatures . . . . .	50
7.1.1	Definition . . . . .	50
7.1.2	Construction and Analysis . . . . .	52
7.2	Incrementally Verifiable Computation . . . . .	54
7.2.1	Definition . . . . .	54
7.2.2	Construction and Analysis . . . . .	55
<b>A</b>	<b>Homomorphic Encryption with Local Compression Compiler</b>	<b>64</b>

# 1 Introduction

Succinct non-interactive arguments (SNARGs) and arguments of knowledge (SNARKs) are not only objects of great importance in the theory of cryptographic proofs, but are also revolutionizing practical applications such as blockchains and cryptocurrencies [But21]. A vigorous and productive line of research has resulted in several constructions of SNARGs and SNARKs for NP (e.g., [Mic94, Gro10, Lip12, BCCT12, GGPR13, BCCT13, BCI<sup>+</sup>13], to name a few). However, despite decades of research, all constructions of SNARGs for general NP languages have relied on either the random oracle model or on non-falsifiable knowledge assumptions. While there are some limited negative results on constructing SNARGs in the standard model under falsifiable cryptographic assumptions [GW11], the possibility of achieving them remains tantalizing, indeed a holy grail of this line of research.

Meanwhile, a steady stream of results, starting from [KRR13, KRR14] has showed us how to achieve SNARGs in the standard model for increasingly more expressive classes of computations. The initial results in this direction compiled multi-prover interactive proofs into *privately verifiable* SNARGs [KRR13, KRR14, BHK17, BKK<sup>+</sup>18], resulting in privately verifiable SNARGs for P (and even for NTISP, the class of non-deterministic time and space bounded computations). More recent incarnations [KPY19, KPY20] of this line of work have resulted in *publicly verifiable* SNARGs for P (and NTISP) using hardness assumptions on groups that support bilinear maps. An alternate, seemingly completely different, route to publicly verifiable SNARGs has proceeded by the round-compression of interactive proofs via the Fiat-Shamir paradigm [FS87, CCH<sup>+</sup>19, JKKZ21]. Following this line of research, the recent breakthrough work of Choudhuri, Jain, and Jin [CJJ21a] constructed a SNARG for P, under the learning with errors (LWE) assumption; and [KVZ21] achieved a SNARG for NTISP from the LWE assumption. The key ingredient in their constructions is a SNARG for batch-NP computations, also referred to as a batch argument or BARG.

BARGs are the central theme in our work. They have recently emerged as a powerful tool for constructing expressive SNARGs for various classes of computations. A BARG is a proof system for  $k$  NP statements where the size of the proof (resp. the verification time) is proportional to the size of a *single instance-witness pair* (resp. the time for a single NP verification). Indeed, [KVZ21, CJJ21a] showed a “BARG-to-SNARG compiler” that uses any BARG to construct a SNARG for P and one for NTISP. Together with the construction of BARGs from LWE [CJJ21a], we have SNARGs for P and NTISP from the LWE assumption. Together with the construction of BARGs from bilinear maps [KPY19, WW22, KLVW22], and from a combination of the decisional Diffie-Hellman (DDH) and quadratic residuosity (QR) assumptions [CJJ21b, KLVW22], the aforementioned BARG-to-SNARG compiler [CJJ21a, KVZ21] gives us SNARGs for P and NTISP from the corresponding assumptions. Batch proofs have also been studied in the statistical soundness setting; see, e.g., [RRR18, RR20]. In summary, the study of batch proofs and arguments shed new light on the long-running quest to construct succinct non-interactive arguments for NP.

**Our work.** In this paper, we advance the study of batch arguments for NP in several directions.

As our main technical contribution, we construct the first *rate-1* BARG in the standard model where the size of the proof for  $k$  NP statements is  $m + \text{poly}(\lambda, \log k)$ , where  $m$  is the length of a single witness and  $\lambda$  is the security parameter. That is, the ratio between the length of the proof to the length of a single witness is asymptotically 1. In all prior work [CJJ21a, CJJ21b, KVZ21, WW22], the BARG proof had a multiplicative overhead in the security parameter (whereas our overhead is additive). Short of attaining the holy grail of constructing a full-fledged SNARG for NP (with proof

length sublinear in the witness length), this is the optimal size one can achieve for BARGs. The soundness of our construction relies on the learning with errors (LWE) assumption.

Our BARG scheme has the following two additional properties (similar to the one from [CJJ21a]). First, it is *somewhere extractable* (we call such schemes **seBARG**) in the sense that given an appropriate trapdoor, one can extract a single witness from the BARG proof. In addition, it is an *index seBARG*, which means that if the  $k$  instances can be generated by a size- $s$  circuit that on input  $i$  outputs the  $i$ -th instance (where  $s$  may be significantly smaller than  $k$ ) then the verification time grows with  $s$  (and is otherwise independent of  $k$ ).<sup>1</sup>

On the way to our construction, we define and construct a *fully-local somewhere extractable hash function* (fISEH) family, a significant generalization of the notion of somewhere statistically binding (SSB) hashing defined by Hubáček and Wichs [HW15]. In SSB hashing, the hash function description specifies a (hidden) set of  $m$  input locations where the hash output is statistically binding. It is not hard to see that this implies the hash output is at least  $m$  bits long. Indeed, in all prior constructions [HW15, OPWW15a], the hash output had length  $m \cdot \text{poly}(\lambda)$ . Our construction improves on this in several ways. First, our construction is *rate-1*, in the sense that the hash output consists of  $m + \text{poly}(\lambda)$  bits. Secondly, it is *extractable* in the sense that given a trapdoor associated to the hash function, one can extract the  $m$  statistically bound bits efficiently<sup>2</sup>; and finally, it is *fully-locally openable* in the sense that the local opening of each bit has size  $\text{poly}(\lambda)$ , *independent of  $m$* . In contrast, the Hubáček-Wichs [HW15] notion of SSB hashing required a weaker version of local opening: the size of the local opening for a single bit could grow polynomially with  $m$ . A minute of thought reveals that this requirement is highly non-trivial: the local opening needs to be shorter than the hash function output which, as we observed above, grows with  $m$ . We view this primitive and the construction thereof to be as important as the result itself, and elaborate on it in Sections 1.2 and 2.3. Indeed, this primitive has already been used in [KLVW22] to compile any semi-succinct BARG scheme (where the BARG proof of  $k$  NP statements has size sublinear in  $k$ ) into a (succinct) BARG scheme.

We show a number of applications of our main result. We show a construction of multi-hop **seBARGs** from LWE; a construction of multi-hop aggregate signatures from LWE (in the standard model); and a construction of incrementally verifiable computation for all deterministic computations from LWE. Previously, such results were known only in idealized models or under non-standard knowledge assumptions, with the exception of aggregate signatures which was known assuming indistinguishability obfuscation (iO) and the RSA assumption [HKW15].<sup>3</sup> We describe our contributions in more detail below.

## 1.1 Our Main Technical Result: Rate-1 seBARG for NP

Our main technical contribution is the construction of a *rate-1 seBARG* scheme. A somewhere extractable BARG scheme (**seBARG**) consists of a setup algorithm  $\text{Gen}$ ; a prover algorithm  $\mathcal{P}$ ; and a verifier algorithm  $\mathcal{V}$ . The setup algorithm  $\text{Gen}$  is given a security parameter  $1^\lambda$ , a parameter  $k$  (which specifies the number of instances in a batch NP statement), an instance size  $n$ , and an index  $i \in [k]$  (which specifies which witness should be extractable). It generates a common reference string

<sup>1</sup>From now on, when we refer to an **seBARG**, we always mean that it is an index **seBARG**.

<sup>2</sup>In fact, our construction turns out to have the stronger property of *local extraction*: that is, the time to extract a single bit is  $\text{poly}(\lambda)$ , independent of  $m$ . However, we do not need this property for our constructions, and hence do not pursue it any further.

<sup>3</sup>We note that the scheme presented in [HKW15] is only selectively secure, whereas our scheme is adaptively secure.

crs and a trapdoor  $\text{td}$ . The prover algorithm  $\mathcal{P}$  is given the crs and a set of instances  $x_1, \dots, x_k \in \mathcal{L}$  together with a set of corresponding witnesses  $w_1, \dots, w_k$ , and generates a proof  $\pi$ . Finally, the verification algorithm  $\mathcal{V}$  is given a crs, a set of instances  $x_1, \dots, x_k$ , and a proof  $\pi$ , and outputs 0/1 (indicating accept or reject).

A rate-1 seBARG scheme has the following properties (in addition to the usual notion of completeness).

1. *Semi-adaptive soundness.* For any index  $i \in [k]$  and  $\text{crs} \leftarrow \text{Gen}(1^\lambda, k, n, i)$ , no probabilistic polynomial-time adversary can generate a tuple of  $k$  NP statements  $x_1, \dots, x_k \in \{0, 1\}^n$  together with a proof  $\pi^*$ , where  $x_i \notin \mathcal{L}$  and yet, the verifier accepts  $\pi^*$ . *Semi-adaptivity* here refers to the fact that the adversary picks  $i$  before seeing the common reference string.
2. *Somewhere proof-of-knowledge.* This is a strengthening of semi-adaptive soundness which requires an probabilistic polynomial-time extractor algorithm that, given a trapdoor for the crs,  $k$  NP statements, and an accepting proof  $\pi^*$ , extracts a witness  $w_i$  for the statement  $x_i$ .
3. *Index hiding.* The common reference string crs should hide the index  $i \in [k]$  (which specifies which witness is extractable).
4. *Efficiency.* The algorithms  $\text{Gen}, \mathcal{P}$  and  $\mathcal{V}$  are all probabilistic polynomial-time, and the proof  $\pi$  has size  $m + \text{poly}(\lambda, \log k) = m + \text{poly}(\lambda)$ . Moreover, if the instances  $x_1, \dots, x_k$  can be efficiently generated by a size- $s$  circuit that on input  $i$  outputs  $x_i$ , then the run-time of  $\mathcal{V}$  is  $\text{poly}(s, m, \lambda)$ , as opposed to the potentially much larger  $\text{poly}(n, k, m, \lambda)$ .

We note that, as remarked before, all known constructions of BARGs and seBARGs [CJJ21a, CJJ21b, WW22] have an inverse polynomial (in  $\lambda$ ) rate. Our main result is the following.

**Theorem 1.1** (Informal). *Under the LWE assumption, for every  $\mathcal{L} \in \text{NP}$ , there exists a rate-1 seBARG scheme  $(\text{Gen}, \mathcal{P}, \mathcal{V})$ .*

We remark that our construction is more general: given a SNARG for an NP language with proofs of size  $\ell(m)$  for statements with  $m$ -bit long witnesses, our construction produces an seBARG with proofs of size  $\ell(m) + \text{poly}(\lambda)$  for  $k \leq 2^\lambda$  statements. Special-purpose NP languages with such non-trivial SNARGs exist, e.g. for the non-deterministic class NTISP. Our construction then “lifts” them into somewhere extractable batch arguments with similarly short proofs. We also note that short of constructing a non-trivial SNARG for NP, rate-1 seBARGs are the best one can do. We refer the reader to Section 3.2 for details.

## 1.2 Our Main Tool: Fully-Local Somewhere Extractable Hash (SEH) Families

The main technical tool that we define, construct, and use is a rate-1 fully-local somewhere extractable hash (fISEH) function.

Collision-resistant hash functions compress their input, therefore by definition lose information about it. Hubáček and Wichs [HW15] ask if there is a hash function that is nevertheless guaranteed to preserve *some* information about its input. For example, letting the input  $x$  be an  $n$ -bit string, we may wish to design a hash function  $h_I$  so that  $h_I(x)$  remembers  $x_I = (x_i)_{i \in I}$  for some subset  $I \subseteq [n]$ . That is, for every  $x, x'$  such that  $x_I \neq x'_I$ ,  $h_I(x) \neq h_I(x')$ . In this sense, the hash function  $h_I$  is statistically binding on the locations  $I \subseteq [n]$ , and hence such a hash family is called a *somewhere*

*statistically binding* (SSB) hash function. It is not hard to see that for this to happen, the hash output must have size at least  $|I|$ . As stated, however, this is trivial to build:  $h_I$  could simply output  $x_I$ . To be non-trivial (and indeed, useful), an SSB hash function needs to have an additional *hiding* property, namely the descriptions of the hash functions  $h_I$  and  $h_{I'}$  should be computationally indistinguishable whenever  $|I|=|I'|$ . Hubáček and Wichs [HW15] show how to construct an SSB hash function family using a leveled fully homomorphic encryption (FHE) scheme, thus relying on the LWE assumption. Subsequent work showed how to realize SSB hash functions from a wider class of assumptions [OPWW15a].

Our notion of a *rate-1 fully-local somewhere extractable hash function* (fISEH) family is a significant generalization of SSB hashing: it is *rate-1*, *extractable*, and *fully-locally openable*.

For a reader familiar with the Hubáček-Wichs construction, it is not hard to see that their construction is *somewhere extractable*, that is, given a trapdoor associated with the hash function, one can extract the statistically bound value. In the Hubáček-Wichs construction, the trapdoor is simply the secret key of the FHE scheme. Their construction also has a *local opening* property, that is, there is an opening algorithm that, given  $(x, i)$  and a description of the hash function, outputs  $x_i$  together with a short “certificate” that certifies the correctness of  $x_i$ . The Hubáček-Wichs construction is a tree-based hash function, similar to a Merkle hash [Mer88]. Thus, an opening consists of all the hash values on the path in the tree from the leaf node  $i$  to the root, together with the hash values of all their siblings. The fact that each hash value is of size at least  $|I|$  implies that the size of the opening is at least  $|I|$ . We refer to a hash family that satisfies these properties as an SEH family. Finally, the Hubáček-Wichs construction achieves *rate-1*, that is, the size of the hash output is  $|I| + \text{poly}(\lambda)$ , if one uses a rate-1 FHE scheme such as the one from [BDGM19, GH19a, DGI<sup>+</sup>19a].

A *rate-1 fully-local SEH* family is a rate-1 SEH family with the crucial additional property of full locality. That is, the size of a local opening does not grow with  $|I|$ , the number of bits that we are statistically bound on. In other words, one should be able to open any bit of the input using an opening of size  $\text{poly}(\lambda)$  bits, independent of  $|I|$ . Additionally, verification of this opening should be possible in time  $\text{poly}(\lambda)$  as well. In other words, verification should take time smaller than the hash value itself! A minute of thought reveals that one has to significantly depart from the Merkle tree paradigm to construct such a hash function. Indeed, the main novelty and technical contribution of our construction is in building such a seemingly paradoxical hash function with *fully* local opening and verification algorithms. We refer the reader to Section 2.3 for details.

**Theorem 1.2** (Informal). *Under the LWE assumption, there exists a rate-1 fully-local SEH family.*

### 1.3 Applications of seBARGs

We show two ways to strengthen seBARGs, resulting in the new notions of *multi-hop seBARG* and *hashed* (multi-hop) seBARG. We then show how to use rate-1 seBARGs to achieve these stronger notions. Finally, we show two applications of these stronger notions: constructing a multi-hop aggregate signature scheme in the standard model, and constructing an incrementally verifiable computation scheme in the standard model. All schemes are secure under the LWE assumption. We describe these contributions in more detail below.

**Multi-Hop seBARGs.** The ability to compose proofs enables mutually distrustful parties to perform distributed computations in a verifiable manner, and is especially important in decentralized applications such as blockchains. The importance of proof composition was in fact already realized

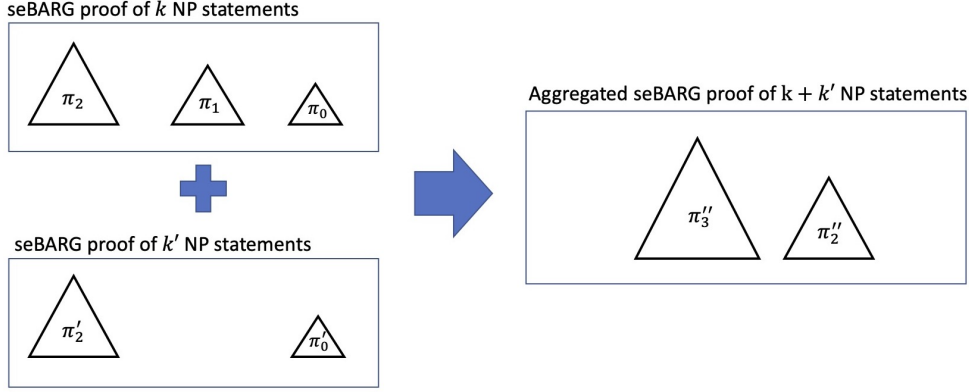


Figure 1: Each proof  $\pi_i$  (resp.  $\pi'_i$ ) is an aggregation of  $2^i$  many NP statements. A proof of  $k$  NP statements is thus a collection of seBARG proofs  $(\pi_0, \dots, \pi_{\log k})$  where each  $\pi_i$  is a proof of  $2^i$  statements (if the  $i^{\text{th}}$  digit in the binary expansion of  $k$  is 1) or empty (if the  $i^{\text{th}}$  digit is 0). To aggregate two such proofs (for  $k = 7$  and  $k = 5$  NP statements respectively, as illustrated above on the left), we start by combining  $\pi_0$  and  $\pi'_0$  into a proof  $\bar{\pi}_1$  that certifies the truth of two NP statements. We then aggregate  $\pi_1$  and  $\bar{\pi}_1$  into a proof  $\pi''_2$  which certifies the truth of four NP statements. Finally, we aggregate  $\pi_2$  and  $\pi''_2$  into a proof  $\pi''_3$  which certifies the truth of eight NP statements. The final proof consists of  $(\pi''_2, \pi''_3)$ . Each proof  $\pi_i$  is a result of at most  $i \leq \log k$  hops of aggregation.

in Micali’s original work introducing SNARGs [Mic93], and has been extensively studied in the last two decades [Val08, CT10, BCCT13, BCTV14, COS20, KPY20, BCMS20, BCL<sup>+</sup>21], leading to powerful primitives such as incrementally verifiable computation (IVC) [Val08] and proof-carrying Data (PCD) [CT10], and to many applications, such as enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], PPAD hardness [Val08, BPR15, KPY20], and succinct blockchains [KB20, BMRS20, CCDW20].

We ask whether it is possible to compose BARG proofs. That is, given BARG proofs  $\pi_1, \dots, \pi_k$  of  $k$  batches of NP statements  $\mathbf{x}_1 = (x_{1,1}, \dots, x_{1,\ell_1}), \dots, \mathbf{x}_k = (x_{k,1}, \dots, x_{k,\ell_k})$ , can we create a BARG proof  $\pi$  for the collection of these NP statements

$$\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_k) = (x_{1,1}, \dots, x_{1,\ell_1}, \dots, x_{k,1}, \dots, x_{k,\ell_k})$$

are true? Furthermore, the system should support an unbounded polynomial number of such iterative compositions without the proof size or the verification time growing by too much. We call such a system a multi-hop batch argument for NP, or a multi-hop BARG for short. If the system supports somewhere extraction, appropriately defined, it is called a multi-hop seBARG.

It is natural to try to realize a multi-hop BARG by BARGing many BARG proofs. In the example above, one could try to produce a BARG proof that there exist BARG proofs  $\pi_1, \dots, \pi_k$  certifying the truth of the collections of NP statements  $\mathbf{x}_1, \dots, \mathbf{x}_k$  respectively. The first issue with this approach is that each composition increases the length of the proofs. For example, if we use a BARG system where the proof length has a multiplicative overhead of  $\text{poly}(\lambda)$  (over the witness length) as is the case for all known BARG schemes [KPY19, CJJ21a, CJJ21b, WW22], then the size of the proofs after  $B$  hops grows to  $\lambda^{\Omega(B)}$ . By playing with the security parameter, one can make the number of hops slightly super-constant (see, e.g., [KPY19, KPY20]), but that hits the limit of what is possible.

Even if we had a constant-rate BARG (with rate  $> 1$ ), we could handle at most a logarithmic (in the security parameter) number of hops. The second issue is one of proving soundness: the most direct way of proving soundness of the composed proof system is through extraction.

Rate-1 somewhere extractable BARGs (seBARGs) solve both these problems, giving us the following theorem.

**Theorem 1.3** (Informal). *Under the LWE assumption, there exists a multi-hop seBARG for NP statements with  $m$ -bit witnesses where the size of the proof after  $B$  hops is  $m + B \cdot \text{poly}(\lambda)$ .*

Our proof of this theorem proceeds by showing a way to convert *any* rate-1 seBARG scheme into a multi-hop seBARG. We obtain the theorem by using our rate-1 seBARG from Theorem 1.1 which we construct under LWE. Each hop incurs an *additive* increase of  $\text{poly}(\lambda)$  to the proof length, and hence the total size after  $B$  hops is a modest  $m + B \cdot \text{poly}(\lambda)$  bits. Our construction does not require the batch size or the instance length to be fixed in advance, relying on the fact that the run-time of our rate-1 seBARG setup algorithm grows only poly-logarithmically with the batch size and the instance length.

Finally, we observe that while the dependence on the number of hops may seem limiting, a simple extension of our construction allows us to batch an *arbitrary* polynomial (and even more, upto an exponential) number of NP proofs. To do so, we maintain the invariant that the batched proof at any point corresponding to  $k$  NP statements consists of a sequence of seBARG proofs  $(\pi_0, \pi_1, \dots, \pi_\ell)$  with  $\ell \leq \lfloor \log k \rfloor$ , where  $\pi_i$  is either empty or a batched proof corresponding to exactly  $2^i$  NP statements. It is not hard to see that two such batched proofs  $(\pi_0, \dots, \pi_\ell)$  and  $(\pi'_0, \dots, \pi'_\ell)$  can be combined together into a proof  $(\pi''_0, \dots, \pi''_\ell)$  maintaining the invariant. When batching a total of  $k$  NP statements, our batched proof consists of at most  $\log k$  multi-hop seBARG proofs, where each proof certifies the correctness of  $2^i$  statements for some unique  $i \in [\log k]$ . Each of these proofs is a result of at most  $\log k$  hops of aggregation and thus has size  $m + (\log k) \cdot \text{poly}(\lambda)$ . Thus, the total proof size grows with  $O(\log^2 k)$  (ignoring the dependence on the witness size  $m$  and the security parameter  $\lambda$ ). See Figure 1 for an illustration.

We refer the reader to Section 6 for more details on the multi-hop seBARG definition and construction.

**Multi-Hop Hashed seBARGs.** To realize some of our applications, such as incrementally verifiable computation, we need a further generalization of (multi-hop) seBARGs, in which the verifier is given only a (somewhere extractable) hash  $\mathbf{v}$  of the *instances*, rather than all the instances in the clear. The somewhere argument of knowledge property now states that if a proof  $\pi$  verifies with respect to a hash value  $\mathbf{v}$ , then we can extract a valid witness  $\omega$  from  $\pi$  for the instance  $x$  which is extractable from  $\mathbf{v}$ . We call this a (multi-hop) hashed seBARG. A hashed seBARG is related to, but different from, an index seBARG: in the latter, there is a compressed representation of the instances from which each instance can be recovered quickly, whereas in the former, the instances can be arbitrary without necessarily a short representation. We believe that the notion of a hashed (multi-hop) seBARG is of independent interest.

We show how to construct a hashed multi-hop seBARG from a rate-1 seBARG. The hashed multi-hop seBARG proof combiner takes as input, instead of  $k$  batches of statements,  $k$  hash values  $\mathbf{v}_1, \dots, \mathbf{v}_k$  of depth  $d$ , and corresponding proofs  $\pi_1, \dots, \pi_k$  and outputs a hash value  $\mathbf{v}$  of depth  $d + 1$  and corresponding proof  $\pi$ . The construction is similar to the regular multi-hop seBARG, except that now when aggregating, every witness includes a hash value  $\mathbf{v}_i$  and an opening  $\rho_i$  of  $\mathbf{v}_i$  w.r.t.  $\mathbf{v}$ ,



in addition to the proof  $\pi_i$ . In the initial aggregation step, the witness is simply an NP witness  $w_i$ , and  $v_i$  is simply  $x_i$ . In later aggregations, the witness becomes a seBARG proof  $\pi_i$  corresponding to a hash value  $v_i$ . We rely on the somewhere extraction property of the hash to ensure that the instance that we recursively extract from the proof (which goes along with the witness  $\omega$  that we recursively extract from the proof), matches the instance  $x$  that we recursively extract from the hash value.

**Theorem 1.4** (Informal). *Under the LWE assumption, there exists a hashed multi-hop seBARG for NP statements with  $n$ -bit instances and  $m$ -bit witnesses where the size of the proof after  $B$  hops is  $m + n \cdot B \cdot \text{poly}(\lambda)$ .*

We refer the reader to Section 6.3 for more details on hashed (multi-hop) seBARG.

**Multi-Hop Aggregate Signature Schemes from LWE.** As an immediate application of our multi-hop seBARG, we construct a multi-hop aggregate signature scheme.

Aggregate signatures were introduced by Boneh, Gentry, Lynn, and Shacham [BGLS03] to enable the compression of a sequence of signatures  $\sigma_1, \dots, \sigma_k$ , where each  $\sigma_i$  is a signature of an arbitrary message  $m_i$  w.r.t. an arbitrary verification key  $\text{vk}_i$ , into a single *aggregated* signature  $\hat{\sigma}$  whose size is independent of  $k$ . While the original motivation for aggregate signatures was to compress certificate chains and to reduce cryptographic overhead in secure BGP, the notion has recently found a great deal of practical interest in the context of blockchains where they provide tangible savings in communication and space. Multi-hop aggregate signatures require the ability to aggregate several aggregate signatures, similar to multi-hop seBARGs.

The original construction of aggregate signatures [BGLS03] relied on bilinear maps and was proven secure in the random oracle model. A more recent construction [HKW15] uses indistinguishability obfuscation and the RSA assumption, and constructs an aggregate signature scheme secure in the standard model. Finally, it is a folklore observation that SNARKs for NP immediately give us an aggregate signature scheme. All these schemes support multi-hop aggregation. Thus, with the exception of [HKW15], all constructions of aggregate signature schemes rely either on random oracles, or on specialized knowledge-type assumptions.

In this work, we observe that our construction of multi-hop seBARGs immediately gives us a multi-hop aggregate signature scheme (in the standard model) secure under the LWE assumption.

**Theorem 1.5** (Informal). *Under the LWE assumption, there exists a multi-hop aggregate signature scheme in the standard model.*

Finally, we note that the “seBARG lens” gives us multi-hop aggregate signatures with several additional desirable properties. For example:

1. *Universal Aggregation:* The goal of universal signature aggregation [HKW15], is to be able to aggregate a collection of signatures produced from *any* signing algorithm (as long as the description of the verification algorithm is fixed). Universal signature aggregation allows real-world systems to continue to use existing signature schemes and public-key infrastructure, while supporting signature aggregation. The generality of the seBARG methodology lets us add the signature aggregation feature to any signature scheme by including the description of the verification algorithm as part of the instance. While [HKW15] and the SNARK-based construction give us universal signature aggregation, [BGLS03] does not.

2. *Local Verifiability*: A locally verifiable aggregate signature scheme [GV22] allows a verifier to check, given an aggregate signature  $\hat{\sigma}$  and a small advice that can be computed from all the  $(m_i, vk_i)$  pairs, whether a particular message  $m$  is in the aggregated set  $(m_1, \dots, m_k)$ , *without having to know the entire list of messages*. Indeed, the signature verifier runs in time sublinear in  $k$ . We note that using a hashed (multi-hop) seBARG scheme, we can construct a (multi-hop) locally verifiable aggregate signature scheme by having the small advice be an opening of the hash value.

We refer the reader to Section 7.1 for more details. While our constructions are not concretely efficient (as compared to, e.g., [BGLS03]), we believe that a refinement of the seBARG lens can potentially give us a concretely efficient LWE-based aggregate signature scheme. We leave the exploration of this line of thought for future work.

**Incrementally Verifiable Computation from LWE.** Consider the computation of a Turing machine  $\mathcal{M}$  on an input  $x$ , a computation so long that no single person can finish it all by herself. Each person thus takes the intermediate state resulting from a partial computation, a Turing machine configuration  $\text{conf}_t$  at time  $t$ , and updates it by running  $\ell$  more steps to get a configuration  $\text{conf}_{t+\ell}$ . To certify that the computation has been done correctly, one needs to do more. Given a succinct proof  $\pi_t$  certifying that the  $t$ -th configuration is  $\text{conf}_t$ , one needs to efficiently compute a new succinct proof  $\pi_{t+\ell}$  certifying that the  $(t + \ell)$ -th configuration is  $\text{conf}_{t+\ell}$ . Importantly, the time to generate  $(\text{conf}_{t+\ell}, \pi_{t+\ell})$  should depend only on  $\ell$ , and should be independent of  $t$ . In particular, the length of the proofs does not grow with  $t$ ; at worst, we allow a poly-logarithmic in  $t$  growth. This is the notion of incrementally verifiable computation (IVC), a notion of great interest in settings where long ongoing computations are performed by a distributed network of mutually distrusting parties [Mic94, Val08, BCCT13].

All known constructions of IVC use full-fledged SNARKs for NP as a building block, and thus rely on non-falsifiable knowledge assumptions. Kalai, Paneth and Yang [KPY20] constructed a weak form of IVC for deterministic time- $T$  computations with proofs of size  $\text{poly}(\lambda, T^\epsilon)$  for any constant (or even slightly sub-constant)  $\epsilon > 0$ , assuming the hardness of problems on elliptic curves that support bilinear maps. Roughly speaking, in their construction,  $\pi_{t+\ell}$  simply consists of  $(\pi_t, \pi')$  where  $\pi'$  certifies the correctness of the computation from  $\text{cf}_t$  to  $\text{cf}_{t+\ell}$ , and after  $p(\lambda)$  many succinct proofs are accumulated (where  $\lambda$  is the security parameter), they are combined into a single succinct proof (using a seBARG), where each such combining step incurs a multiplicative blowup of  $q(\lambda) \ll p(\lambda)$  to the proof size. Therefore, if this combining step is applied  $B$  times then the resulting proof increases by a factor of  $q(\lambda)^B$  which allows for only a constant number of combination steps. Setting parameters appropriately, this results in a somewhat succinct proof of size  $T^\epsilon$  for any constant  $\epsilon > 0$ , where  $T$  is the run-time of the computation. (We remark that a careful balancing act with the security parameter gives them slightly more, namely a slightly sub-constant  $\epsilon$ , but they are inherently limited by this exponential growth of the proof in the depth  $B$ .)

We note that our rate-1 seBARG scheme allows us to do this combining step while paying only an *additive*  $\text{poly}(\lambda)$  blowup. In particular, using our seBARG, we can apply this combining step  $B$  times while incurring only an additive blowup of  $B \cdot \text{poly}(\lambda)$ , and while relying only on the LWE assumption. This allows us to obtain an IVC scheme for *any* deterministic computation, even ones in EXP (e.g., by using a tree-like structure in which the depth, and thus the blowup, is only *logarithmic* in the run-time of the computation).

**Theorem 1.6** (Informal). *Under the LWE assumption, there exists an incrementally verifiable computation scheme for any deterministic computation, where the proofs grow poly-logarithmically with the run-time of the computation.*

Our construction of an IVC scheme uses (hashed) multi-hop seBARGs. In the first hop, we batch  $k$  instances, where the  $i$ -th instance is  $(x, \text{cf}_{i-1}, \text{cf}_i)$ . The witness for this statement is empty since verifying an instance can be done efficiently. It should be possible to verify the batched proof given only the initial configuration  $\text{cf}_0$  and the last configuration computed so far  $\text{cf}_k$ , without knowing the intermediate configurations. This is crucial to obtain our desired efficiency gain. This is where our notion of a *hashed* multi-hop seBARG comes in. Using a hashed multi-hop seBARG, we can verify proofs without knowing all the corresponding instances, rather only their (somewhere statistically binding) hashes.

Continuing along these lines, we reach the second stage where we have  $k$  proofs  $\pi_1, \dots, \pi_k$  and  $k$  hash values  $\mathbf{v}_1, \dots, \mathbf{v}_k$ , where  $\pi_i$  certifies that  $\mathbf{v}_i$  is a hash of consecutive configurations starting with  $\text{cf}_{k(i-1)+1}$  and ending with  $\text{cf}_{ki}$ . We can feed these into the hashed multi-hop seBARG proof combiner and get an aggregated proof  $\pi$ , but we run into an issue along the “boundaries:” there is no guarantee that  $\text{cf}_{k(i-1)}$  actually transitions to  $\text{cf}_{k(i-1)+1}$  for  $i \in \{2, \dots, k\}$ .

We handle this by overlapping the batches that we aggregate. In the first hop, we actually batch  $2k$  instances at a time, so the second half of each batch overlaps with the first half of the next batch: The proof  $\pi_i$  actually certifies that  $(\mathbf{v}_{i-1}, \mathbf{v}_i)$  is a hash of consecutive configurations starting with  $\text{cf}_{k(i-2)+1}$  and ending with  $\text{cf}_{ki}$ , so it indeed certifies that  $\text{cf}_{k(i-1)}$  actually transitions to  $\text{cf}_{k(i-1)+1}$ . This way, the problematic “boundary” of any batch is completely contained within another batch.

We refer the reader to Section 7.2 for more details on the IVC definition and construction.

## 2 Technical Overview

We sketch our main result that takes any seBARG scheme and converts it into a rate-1 seBARG, assuming a rate-1 additive homomorphic encryption scheme with certain properties (which in turn can be constructed based on LWE).

### 2.1 Main Ingredient: Flexible RAM SNARGs with Partial Input Soundness

The central tool that we use repeatedly in our construction is a flexible RAM SNARG with partial input soundness, a primitive that was very recently introduced and constructed in [KLVW22].

**Remark 2.1.** Our work and that of [KLVW22] are intertwined. An earlier (unpublished) version of this work did not use RAM SNARGs, and used *quasi-arguments* [KPY19] instead. Following [KLVW22], we noticed that their notion of *flexible* RAM SNARG *with partial input soundness* can be used to significantly simplify our construction and analysis. It is this simplified version that is presented in this paper.

The notion of a RAM delegation scheme was introduced and constructed in the work of Kalai and Paneth [KP16]. A (publicly-verifiable non-interactive) RAM delegation scheme, is a SNARG for RAM computations where a prover, given a  $\text{crs}$ , a time- $T$  RAM machine  $M$  and an  $n$ -bit input  $x$ , can produce a short,  $\text{poly}(\lambda, \log T)$ -bit, proof that  $y = M(x)$ . Given a  $\text{poly}(\lambda)$ -bit “digest” of the input  $x$ , the verifier runs in time  $\text{poly}(\lambda, \log T)$ , *independent of the input length  $n$* , and either

accepts or rejects the proof. The key difference from a SNARG for  $\mathcal{P}$  is that the prover runtime is proportional to the RAM runtime, but more importantly, that the verifier runtime, given the input digest, is independent of the input length and polylogarithmic in the RAM runtime.

For a formal definition, we refer the reader to Section 3.3.

For our work, the standard notion of a RAM SNARG is insufficient, and we need a RAM SNARG with two additional properties, as was defined and constructed in [KLVW22]: First, the RAM SNARG has to be *flexible* in the sense that any hash family with local opening can be used to digest the input. Second, it has a stronger soundness guarantee known as *partial input soundness*. In prior RAM SNARGs, soundness is guaranteed only if the adversary knows the entire input that is being digested. Partial input soundness guarantees that if the memory is digested using a somewhere extractable hash function that is extractable on a set of coordinates  $S$ , and if the RAM computation only reads coordinates in  $S$ , then soundness holds.

Such a flexible RAM SNARG with partial input soundness was constructed in [KLVW22] from any seBARG scheme (and SEH hash) and was used to boost the succinctness of seBARG proofs.

## 2.2 Our Rate-1 seBARG Scheme

The high level intuition behind our construction is the following. Suppose a prover is given  $x_1, \dots, x_k$  and corresponding witnesses  $\omega_1, \dots, \omega_k$ , and wishes to convince the verifier that  $x_1, \dots, x_k$  are all in the language. The basic idea is to simply hash all the witnesses using a rate-1 SSB hash function, so that the hash value  $v$  is statistically binding on a single witness  $\omega_{i^*}$ . The seBARG proof will consist of  $v$  together with a succinct proof  $\pi$  that  $v$  is obtained by hashing valid witnesses for  $x_1, \dots, x_k$ . Importantly, to ensure that the proof  $(v, \pi)$  is of rate-1, i.e., of length  $m + \text{poly}(\lambda)$  where  $m$  is the length of a single witness, the succinct proof  $\pi$  must be of length  $\text{poly}(\lambda)$ , since  $v$  is already of length  $m + \text{poly}(\lambda)$ .

Constructing such a succinct proof is the main technical challenge and novelty of our rate-1 seBARG construction. Jumping ahead, we note that we do not know how to construct such a succinct proof if  $v$  is computed using any rate-1 SSB hash family, and we need to use a (rate-1) *fully-local* SEH hash family (fISEH), a primitive described in Section 1.2. Constructing a rate-1 fISEH hash family is a grand challenge of its own, and we elaborate on it in Section 2.3 and Section 4.

At first it may seem that this approach, of constructing a succinct proof that  $v$  is a hash of valid witnesses, is doomed to fail since it requires a SNARG for NP, which is currently out of our reach. Yet, on a closer look, we notice that a SNARG for all of NP is not necessary. The reason is that a witness  $\omega_{i^*}$  for  $x_{i^*}$  can be efficiently extracted from the hash value  $v$  given the trapdoor  $\text{td}$ . Indeed it is this hash value that saves the day!

In order to prove that  $v$  is a commitment to valid witnesses, we draw inspiration from the works on RAM delegation (starting with [KP16]), and in particular, we use the recent notion of a *flexible* RAM SNARG *with partial input soundness* [KLVW22]. Recall that a flexible RAM SNARG allows us to digest the memory with any hash family with local opening, and in particular with a fully-local SEH (fISEH) hash family. The partial input soundness property guarantees that if the hash value  $v$  is extractable on  $w_{i^*}$  and the RAM program only touches the memory locations corresponding to  $w_{i^*}$  then soundness holds.

Given such a RAM SNARG scheme, we proceed as follows: in our seBARG scheme, the prover generates for every  $i \in [k]$  a succinct RAM SNARG  $\pi_i$  for the RAM computation that checks that the  $i$ -th witness in memory is a valid witness of  $x_i$ . The size of each such proof grows with the size of a

local opening, and moreover, with the time it takes to verify a local opening. This is precisely where the need for a rate-1 fISEH hash family comes in, as it allows each proof  $\pi_i$  to be of size  $\text{poly}(\lambda)$ .

Note that if the prover would send  $(\pi_1, \dots, \pi_k)$  to the verifier then we could argue semi-adaptive soundness by sampling a hash key where the extractability is on the witness corresponding to the instance  $x_{i^*}$  which is not in the language, and then use the partial input soundness to argue that  $\pi_{i^*}$  would be rejecting with overwhelming probability. The issue is that we want our seBARG to be succinct and hence cannot afford to send all the proofs  $(\pi_1, \dots, \pi_k)$ . Instead we will send a seBARG proof (which need not be rate-1) that for every  $i \in [k]$  there exists a RAM SNARG  $\pi_i$  that is a valid proof w.r.t. the digest  $v$ .

### 2.3 Our Fully-Local SEH Family

We next describe at a high-level the ideas behind our construction of a rate-1 fully-local SEH (fISEH) family.

**Rate-1 SEH family.** As a first step, we focus on achieving the rate-1 property (without the fully-local property). Our construction uses as a building block the SEH family of Hubáček and Wichs [HW15], which is a tree-like construction that can be instantiated using any leveled fully homomorphic encryption (FHE) scheme [BV11, BGV12]. We note that in their construction, the hash key  $hk$  consists of encryptions of the indices  $\{i_1, \dots, i_m\}$  where the hash function is statistically bound, and the hash value  $\text{Hash}(hk, x)$  consists of a bit-by-bit encryption of  $x_{i_1}, \dots, x_{i_m}$ .

To make this construction rate-1, we use a rate-1 levelled FHE [BDGM19, GH19b, DGI<sup>+</sup>19b] as the underlying encryption. We use the fact that such an FHE scheme can convert any set of (evaluated) ciphertexts  $ct_1, \dots, ct_m$ , each encrypting a single bit  $b_i$ , into a ciphertext  $v$  of size  $m + \text{poly}(\lambda)$  encrypting  $(b_1, \dots, b_m)$ . Thus, the idea is to output  $v$  (which is rate-1), as opposed to outputting the bit-wise ciphertexts  $ct_1, \dots, ct_m$ . Unfortunately, by outputting only  $v$  we lose the local opening property. This is fixed as follows.

**Obtaining fully-local opening.** Recall that our goal is to provide a local opening to any bit of the input that can be verified in time  $\text{poly}(\lambda)$  (i.e., significantly less than  $m$ ). Thus, an opening needs to be verified without even reading  $v$ ! To reach this goal, we send in addition to  $v$  a somewhere extractable hash of the ciphertexts  $ct_1, \dots, ct_m$ , denoted by  $h$ . Importantly  $h$  is statistically binding on only one ciphertext  $ct_i$  and hence is of size  $\text{poly}(\lambda)$ .

We use  $v$  to extract the  $m$  coordinates, and use  $h$  to open any desired coordinate. We elaborate on the opening procedure below, and mention that to verify the validity of an opening we use only  $h$  (which is succinct) and do not use  $v$ . This allows us to obtain our fully-local guarantee.

For this approach to work we need a mechanism that checks consistency between  $v$  and  $h$ . Thus our hash value actually contains three components  $(v, h, \pi)$ , where  $\pi$  is a succinct proof that certifies the consistency between  $ct$  and  $h$ . The main technical burden is in constructing such a proof of consistency. Note that to obtain our promised rate-1 construction this proof must be of size  $\text{poly}(\lambda)$ . We defer the description of this proof, and we first discuss the local opening (which is substantially simpler).

**Fully-local opening using  $h$ .** In the construction of Hubáček and Wichs [HW15], an opening to a bit  $b$  consists of the bit  $b$  together with  $m$  openings,  $\rho_1, \dots, \rho_m$ , where each  $\rho_j$  corresponds to the

output ciphertext  $\text{ct}_j$ . Moreover, each  $\rho_j$  is of size  $\text{poly}(\lambda)$  and can be verified in time  $\text{poly}(\lambda)$  given  $\text{ct}_j$ . To obtain full succinctness, rather than sending all these openings  $\rho_1, \dots, \rho_m$  (together with local openings of  $\text{ct}_1, \dots, \text{ct}_m$ ), we give a somewhere extractable BARG proof that for every  $j \in [m]$  there exists a triplet  $(\rho_j, \text{ct}_j, o_j)$  where  $\rho_j$  is a valid opening to  $b$  w.r.t.  $\text{ct}_j$  and that  $o_j$  is a valid opening to  $\text{ct}_j$  w.r.t.  $h$ .

Thus, our construction uses not only the SEH family corresponding to a rate-1 FHE scheme, but also uses a somewhere extractable BARG scheme.

**Proving consistency between  $\text{ct}$  and  $h$ .** The remaining ingredient is the succinct proof  $\pi$  that certifies the consistency between  $v$  and  $h$ . Namely, we need to prove that there exist ciphertexts  $\text{ct}_1, \dots, \text{ct}_m$  that hash to  $h$  and their corresponding rate-1 ciphertext is indeed  $v$ . Note that this is an NP computation and SNARGs for NP are beyond our reach, and is harder than our goal that we started with. Thus, it seems that we are back to square one.

However, we observe that this NP computation has special properties that will allow us to construct a SNARG for it. The initial observation is that the NP witness is hashed (via  $h$ ), and thus one may hope to use RAM delegation here. Unfortunately, the soundness of RAM delegation is not strong enough, since it provides soundness only if the adversary knows the memory that is digested. In our setting the memory is  $\text{ct}_1, \dots, \text{ct}_m$  and we have no means of extracting these ciphertexts from the adversary. What saves the day is that we can extract one ciphertext  $\text{ct}_i$  (since the underlying hash is somewhere extractable).

Indeed, to construct this SNARG we use a RAM SNARG from the recent work of [KLVW22]. This RAM SNARG has two properties that are crucial in our setting. First, it is *flexible* in the sense that the memory can be digested using any hash family with local opening, and in particular using any hash family that is also locally extractable. Second, it has a stronger soundness guarantee, known as *partial input soundness* which guarantees correctness if the RAM computation only reads the memory from locations in  $S \subseteq [N]$  (where  $N$  is the memory length) and the hash function is extractable on the set  $S$ .

At first, it is not clear that this soundness guarantee is helpful since the RAM computation that computes  $v$  from the memory  $(\text{ct}_1, \dots, \text{ct}_m)$  reads the entire memory. Yet, if we use the scheme from [BDGM19] (which is based on LWE) then  $v$  is computed in the following way, which does allow us to make use of this underlying RAM SNARG.

1. First, each ciphertext  $\text{ct}_j$  is expanded into a vector ciphertext, denote by  $\text{ct}'_j$ .
2. Second, all these vector ciphertexts are added (over a finite field) to obtain a ciphertext  $\text{ct}' = \sum_{j=1}^m \text{ct}'_j$ .
3. Finally, each bit of  $v$  is computed separately, and depends only on a few bits from  $\text{ct}'$ .

Note that by the partial input soundness guarantee one can compute a RAM SNARG for each bit of  $\text{ct}'_j$  that certifies its correctness w.r.t.  $h$ , and correctness is guaranteed if  $h$  is extractable on  $\text{ct}_j$ . Denote all the proofs corresponding to  $\text{ct}'_j$  by  $\pi'_j$ .

We append to the hash value a somewhere extractable hash of

$$(\text{ct}'_1, \pi'_1, \dots, \text{ct}'_m, \pi'_m)$$

If Item 2 did not exist and each bit of  $v$  depended only on a few bits  $(\text{ct}'_1, \dots, \text{ct}'_m)$  then we would be done since we could simply add a BARG asserting that each bit of  $v$  is computed correctly, where

each witness is the relevant bits from  $(\text{ct}'_1, \dots, \text{ct}'_m)$  and the corresponding proofs and all their openings.

Unfortunately, Item 2 does exist which causes an additional complication, stemming from the fact that this computation is not local (as opposed to Item 1 and 3 which are local). What saves the day is the fact that the computation in Item 2 is simply addition, which can be computed in a tree-like manner where each node computation is a local computation in the nodes of the layer below. Thus, we add  $\log m$  hash values  $h_1, \dots, h_{\log m}$ , where  $h_i$  is the hash of all the computations at layer  $i$  of the tree along of RAM proofs of correctness, where each computation depends only on a few coordinates of the memory at layer  $i - 1$  of the tree, so we can rely on the partial input soundness guarantee.

To summarize, in our construction of a flSEH family, the hash value consists of the following components:

1. A hash value  $h$ , which is a somewhere extractable hash of  $(\text{ct}_1, \dots, \text{ct}_m)$ , which in turn is the hash value from the construction of [HW15]. The size of  $h$  is  $\text{poly}(\lambda)$ .
2. A rate-1 ciphertext  $v$ , which is a rate-1 version of  $(\text{ct}_1, \dots, \text{ct}_m)$ , of size  $m + \text{poly}(\lambda)$ .
3. A proof of consistency between  $h$  and  $v$  which consists of hash values  $(h', h_0, \dots, h_{\log m})$  together with a seBARG proof. Each of these hash values hashes ciphertexts and proofs of consistency, and the final seBARG proof proves consistency between  $h_{\log m}$  and  $v$  and that all the proofs hashed in  $h_{\log m}$  are accepting.

### 3 Preliminaries

**Notations.** We use PPT to denote probabilistic polynomial-time, and denote the set of all positive integers up to  $n$  as  $[n] := \{1, \dots, n\}$ . Also, we use  $[0, n]$  to denote the set of all non-negative integers up to  $n$ , i.e.  $[0, n] := \{0\} \cup [n]$ . Throughout this paper, unless specified, all polynomials we consider are positive polynomials. 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  $\mathcal{D}$ .

#### 3.1 Somewhere Extractable Hash (SEH) Families

In what follows we recall the definition of a somewhere extractable (SEH) hash family based on prior works [HW15, OPWW15b].

**Syntax.** A somewhere extractable hash family SEH consists of the following polynomial time algorithms:

$\text{Gen}(1^\lambda, N, I) \rightarrow (\text{hk}, \text{td})$ . This is a probabilistic setup algorithm that takes as input a security parameter  $1^\lambda$  in unary, a message length  $N$ , and a subset  $I \subseteq [N]$ . It outputs a hash key  $\text{hk}$  along with trapdoor  $\text{td}$ .

$\text{Hash}(\text{hk}, x) \rightarrow v$ . This is a deterministic algorithm that takes as input a hash key  $\text{hk}$  generated by  $\text{Gen}(1^\lambda, N, I)$  and an input  $x \in \{0, 1\}^N$ , and outputs a hash value  $v \in \{0, 1\}^{I \cdot \text{poly}(\lambda)}$ .

$\text{Open}(\text{hk}, x, j) \rightarrow (b, \rho)$ . This is a deterministic algorithm that takes as input a hash key  $\text{hk}$  generated by  $\text{Gen}(1^\lambda, N, I)$ , an input  $x \in \{0, 1\}^N$  and an index  $j \in [N]$ , and outputs a bit  $b \in \{0, 1\}$  and an opening  $\rho \in \{0, 1\}^{\leq |I| \cdot \text{poly}(\lambda)}$ .

$\text{Verify}(\text{hk}, v, j, b, \rho)$ . This is a deterministic algorithm that takes as input a hash key  $\text{hk}$  generated by  $\text{Gen}(1^\lambda, N, I)$ , a hash value  $v \in \{0, 1\}^{|I| \cdot \text{poly}(\lambda)}$ , an index  $j \in [N]$ , a bit  $b \in \{0, 1\}$  and an opening  $\rho \in \{0, 1\}^{\leq |I| \cdot \text{poly}(\lambda)}$ , and outputs 1 (accept) or 0 (reject).

$\text{Extract}(\text{td}, v, j) \rightarrow u$ . This is a deterministic extraction algorithm that takes as input a trapdoor  $\text{td}$  generated by  $\text{Gen}(1^\lambda, N, I)$ , a hash value  $v$ , and index  $j \in [I]$  and outputs a bit  $u$ .

We sometimes use the notation  $\text{Extract}(\text{td}, v) = (\text{Extract}(\text{td}, v, j))_{j \in [I]}$ .

**Definition 3.1 (SEH).** *A somewhere extractable hash family  $\text{SEH} = (\text{Gen}, \text{Hash}, \text{Open}, \text{Verify}, \text{Extract})$  is required to satisfy the following properties:*

**Efficiency.** *The size of the hash key  $\text{hk}$  and the hash value  $v$  is at most  $|I| \cdot \text{poly}(\lambda)$ .*

**Index hiding.** *For any poly-size adversary  $\mathcal{A}$ , any polynomial  $N = N(\lambda)$ , and any  $I_0, I_1 \subseteq [N]$  such that  $|I_0| = |I_1|$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,*

$$\Pr \left[ \mathcal{A}(\text{hk}) = b : \begin{array}{l} b \leftarrow \{0, 1\} \\ (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda),$$

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

$$\Pr \left[ \begin{array}{l} b = x_j \\ \wedge \text{Verify}(\text{hk}, v, j, b, \rho) = 1 \end{array} : \begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ v = \text{Hash}(\text{hk}, x), \\ (b, \rho) = \text{Open}(\text{hk}, x, j), \end{array} \right] = 1.$$

**Somewhere statistically binding w.r.t. opening.** *For any  $\lambda \in \mathbb{N}$ , any  $N \leq 2^\lambda$ , any subset  $I \subseteq [N]$ , any index  $i^* \in I$ , and any (all powerful) adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,*

$$\Pr \left[ \begin{array}{l} \text{Verify}(\text{hk}, v, i^*, b, \rho) = 1 \\ \wedge b \neq b_{i^*} \end{array} : \begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (v, b, \rho) \leftarrow \mathcal{A}(\text{hk}), \\ (b_i)_{i \in I} = \text{Extract}(\text{td}, v) \end{array} \right] \leq \text{negl}(\lambda).$$

**Extraction correctness.** *For any  $\lambda \in \mathbb{N}$ , any  $N \leq 2^\lambda$ , any subset  $I \subseteq [N]$ , and any  $x \in \{0, 1\}^N$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,*

$$\Pr \left[ (x_i)_{i \in I} \neq \text{Extract}(\text{td}, v, I) : \begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I) \\ v \leftarrow \text{Hash}(\text{hk}, x) \end{array} \right] \leq \text{negl}(\lambda).$$

**Remark 3.1.** Note that the index hiding property and the somewhere statistically binding w.r.t. opening property of a SEH hash family, implies the following property:



**Computational binding w.r.t opening.** For any poly-size adversary  $\mathcal{A}$  any polynomial  $N = N(\lambda)$  and any  $I \subseteq [N]$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \forall b \in \{0, 1\} \\ \text{Verify}(\text{hk}, v, i, b, \rho_b) = 1 \end{array} \quad ; \quad \begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (i, v, \rho_0, \rho_1) \leftarrow \mathcal{A}(\text{hk}) \end{array} \right] \leq \text{negl}(\lambda).$$

**Lemma 3.2.** [KLW22] *Let  $(\text{E.Gen}, \text{Enc}, \text{Dec})$  be any encryption scheme that is either levelled fully homomorphic or rate-1 additively homomorphic. Then there is a somewhere extractable hash family with algorithms  $\text{Gen}, \text{Hash}$  that satisfies the following:*

1. For every  $\lambda \in \mathbb{N}$ , any  $N = N(\lambda) \leq 2^\lambda$ , any  $m \leq N$ , any  $I = \{i_1, \dots, i_m\} \subseteq [N]$ , and any  $(\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I)$ ,

$$\text{hk} = (\text{pk}, \text{hk}_1, \dots, \text{hk}_m)$$

where for every  $j \in [m]$ ,  $\text{hk}_j \in \{0, 1\}^{\text{poly}(\lambda)}$  is an encryption of the index  $i_j \in [N]$  w.r.t.  $\text{pk}$ , and  $\text{td}$  is the secret key corresponding to  $\text{pk}$ .

In particular, if  $I$  consists of  $m$  consecutive indices  $I = \{i + 1, \dots, i + m\}$  then one can set  $\text{hk} = (\text{pk}, \text{hk}_1)$ , and  $(\text{hk}_2, \dots, \text{hk}_m)$  can be computed efficiently from  $\text{hk}$ . In this case,  $\text{Gen}$  runs in time  $\text{poly}(\lambda)$ .

2. There exists a negligible function  $\text{negl}$  such that for any  $\lambda \in \mathbb{N}$ , any  $N = N(\lambda) \leq 2^\lambda$ , any  $m \leq N$ , any  $I = \{i_1, \dots, i_m\} \subseteq [N]$ , any  $j \in [m]$ , and any  $x \in \{0, 1\}^N$ ,

$$\Pr \left[ \begin{array}{l} \text{Dec}(\text{td}, v_j) = x_j \\ (v_1, \dots, v_m) = \text{Hash}(\text{hk}, x) \end{array} \quad ; \quad \begin{array}{l} (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (v_1, \dots, v_m) = \text{Hash}(\text{hk}, x) \end{array} \right] \leq \text{negl}(\lambda).$$

In particular, for every  $j \in [m]$ ,  $v_j \in \{0, 1\}^{\text{poly}(\lambda)}$ .

**Remark 3.2.** In [KLW22] a somewhere extractable hash family was constructed from any rate-1 string OT scheme, which is known to imply a rate-1 additively homomorphic encryption scheme.

### 3.2 Somewhere Extractable Batch Arguments (seBARGs)

**Syntax.** A (publicly verifiable and non-interactive) somewhere extractable batch argument scheme seBARG for an NP language  $\mathcal{L}$  consists of the following polynomial time algorithms:

$\text{Gen}(1^\lambda, k, n, i^*) \rightarrow (\text{crs}, \text{td})$ . This is a probabilistic algorithm that takes as input a security parameter  $1^\lambda$ , number of instances  $k$ , input length  $n$ , and an index  $i^* \in [k]$ . It runs in time at most  $\text{poly}(\lambda, n, \log k)$  and outputs a crs  $\text{crs}$  along with a trapdoor  $\text{td}$ .

$\mathcal{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \rightarrow \pi$ . This is a prover algorithm takes as input a  $\text{crs}$ ,  $k$  instances  $x_1, \dots, x_k$  and corresponding witnesses  $\omega_1, \dots, \omega_k$ , and outputs a proof  $\pi$ .

$\mathcal{V}(\text{crs}, x_1, \dots, x_k, \pi) \rightarrow 0/1$ . The verification algorithm takes as input a  $\text{crs}$ ,  $k$  instances  $x_i$  for  $i \in [k]$ , and a proof  $\pi$ . It outputs a bit to signal whether the proof is valid or not.

**Definition 3.3** (seBARG). *A somewhere-extractable batch argument scheme  $\text{seBARG} = (\text{Gen}, \mathcal{P}, \mathcal{V})$  for  $\mathcal{L}$  is required to satisfy the following properties:*

**Efficiency.** *The size of the CRS and the proof is at most  $\text{poly}(\lambda, \log k, n, m)$ , where  $m$  is the witness length.*

**Completeness.** *For any  $\lambda \in \mathbb{N}$ , and any  $k = k(\lambda)$ ,  $n = n(\lambda)$  of size at most  $2^\lambda$ , any  $k$  instances  $x_1, \dots, x_k \in \mathcal{L}$ , and their corresponding witnesses  $\omega_1, \dots, \omega_k \in \{0, 1\}^m$ , and any index  $i^* \in [k]$ ,*

$$\Pr \left[ \mathcal{V}(\text{crs}, x_1, \dots, x_k, \pi) = 1 : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*), \\ \pi \leftarrow \mathcal{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \end{array} \right] = 1.$$

**Index hiding.** *For any poly-size adversary  $\mathcal{A}$ , any polynomials  $k = k(\lambda)$  and  $n = n(\lambda)$ , and any indices  $i_0, i_i \in [k]$  there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,*

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

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

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, x_1, \dots, x_k, \pi) = 1 \\ \wedge \omega^* \text{ is not a valid witness for } x_{i^*} \in \mathcal{L} \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*) \\ (x_1, \dots, x_k, \pi) = \mathcal{A}(\text{crs}) \\ \omega^* \leftarrow \mathcal{E}(\text{td}, \{x_i\}_{i \in [k]}, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

**Remark 3.3.** We note that the somewhere argument of knowledge property implies the following *semi-adaptive soundness* property which asserts that for any poly-size adversary  $\mathcal{A}$ , any polynomials  $k = k(\lambda)$  and  $n = n(\lambda)$ , and any index  $i^* \in [k]$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, x_1, \dots, x_k, \pi) = 1 \\ \wedge x_{i^*} \notin \mathcal{L} \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*) \\ (x_1, \dots, x_k, \pi) = \mathcal{A}(\text{crs}) \end{array} \right] \leq \text{negl}(\lambda).$$

**Remark 3.4.** In Section 4, we use a seBARG scheme which is somewhere extractable on a set of indices  $I = \{i_1, \dots, i_\ell\}$ , where the size of the CRS and proof and runtime of the verifier are allowed to grow with  $\ell$ . This is trivially achievable using any generic seBARG scheme  $(\text{Gen}, \mathcal{P}, \mathcal{V})$  which is somewhere extractable on one index by creating  $\ell$  separate parallel CRS and corresponding proof segments. The setup algorithm, given  $1^\lambda, k, n, I$ , samples  $\text{crs}_j \leftarrow \text{Gen}(1^\lambda, k, n, i_j)$  for each  $j \in [\ell]$  and outputs  $\text{crs} = (\text{crs}_j)_{j \in [\ell]}$ . The prover, given  $\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k$ , computes  $\pi_j \leftarrow \mathcal{P}(\text{crs}_j, x_1, \dots, x_k, \omega_1, \dots, \omega_k)$  for each  $j \in [\ell]$  and outputs  $\pi = (\pi_j)_{j \in [\ell]}$ . The verifier, given  $\text{crs}, x_1, \dots, x_k, \pi$ , outputs 1 if and only if  $\mathcal{V}(\text{crs}_j, x_1, \dots, x_k, \pi_j)$  for each  $j \in [\ell]$ . The extractor, given  $\text{td}, x_1, \dots, x_k, \pi$ , computes  $\omega_j^* \leftarrow \mathcal{E}(\text{td}, x_1, \dots, x_k, \pi_j)$  for each  $j \in [\ell]$  and outputs  $(\omega_j^*)_{j \in [\ell]}$ .

**Definition 3.4.** *A seBARG scheme  $(\text{Gen}, \mathcal{P}, \mathcal{V})$  is said to be an index seBARG if the run-time of  $\mathcal{V}$  on instances  $x_1, \dots, x_k$ , where  $x_i = (x, i)$  for every  $i \in [k]$ , is at most  $\text{poly}(\lambda, |x|, \log k)$ .*

**Theorem 3.5** ([CJJ21a]). *Assuming the hardness of the Learning with Errors (LWE) problem, there exists an index seBARG scheme with proof length  $|\pi| = m \cdot \text{poly}(\lambda, \log k)$ .*

**Rate-1 BARGs.** In this work, we construct a rate-1 seBARG under the LWE assumption. Moreover, our  $\text{crs}$  grows only poly-logarithmically in  $n$ . Informally, in a rate-1 BARG the proof size is the size of a single witness with only an additive overhead. Formally, we define it as follows:

**Definition 3.6.** A seBARG scheme  $(\text{Gen}, \mathcal{P}, \mathcal{V})$  is said to be rate-1 if the following two conditions are satisfied:

1.  $\text{Gen}$  runs in time  $\text{poly}(\lambda, \log n, \log k)$ ,<sup>4</sup> and outputs  $\text{crs}$  of size  $|\text{crs}| = \text{poly}(\lambda, \log k, \log n)$ .
2. The proof generated by  $\mathcal{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k)$  is of length  $m + \text{poly}(\lambda, \log k)$ .

### 3.3 Flexible RAM SNARGs with Partial Input Soundness

In this work we use a flexible RAM SNARG with partial input soundness, as defined in [KLVW22]. Such a RAM SNARG is for RAM computations that only read from memory (and do not write). However, it is flexible with respect to the hash family used to digest the memory, and works with any hash family with local opening. Jumping ahead, the reason we need this flexibility is that our rate-1 seBARG construction in Section 5, uses a RAM SNARG which digests its memory using a rate-1 fully-local somewhere extractable hash function (defined and constructed in Section 4).

Importantly, this RAM SNARG achieves a soundness guarantee known as *partial input soundness*, which is stronger than the soundness achieved in the recent works of [KPY19, CJJ21a]. In what follows, for the sake of simplicity, we define a flexible RAM SNARG, where the flexibility is only with respect to somewhere extractable hash family (see Section 3.1), as opposed to any hash family with local opening.

**Syntax.** Let  $\mathcal{R}$  be a RAM machine. A *flexible* (publicly verifiable and non-interactive) RAM SNARG for  $\mathcal{R}$  is associated with a somewhere extractable hash family (Section 3.1)

$$\text{SEH} = (\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify}, \text{SEH.Extract}),$$

and consists of the following polynomial time algorithms:

$\text{Gen}(1^\lambda, T) \rightarrow \text{crs}$ . This is a probabilistic setup algorithm that takes as input a security parameter  $1^\lambda$  and a time bound  $T$ , and outputs a common reference string  $\text{crs}$ .

$\text{Digest}(\text{hk}, x) \rightarrow \mathbf{v}$ . This is a deterministic polynomial-time algorithm that takes as input an SEH hash key  $\text{hk}$ , generated by  $\text{SEH.Gen}(1^\lambda, N, I)$  for some  $N \in \mathbb{N}$ , and a bit string  $x \in \{0, 1\}^N$ , and outputs the digest  $\mathbf{v} = \text{SEH.Hash}(\text{hk}, x)$  of size  $|I| \cdot \text{poly}(\lambda)$ .

$\mathcal{P}(\text{crs}, \text{hk}, x_{\text{imp}}, x_{\text{exp}}) \rightarrow (b, \pi)$ . This is a deterministic prover that takes as input a  $\text{crs}$ , a hash key  $\text{hk}$ , and an input  $x = (x_{\text{imp}}, x_{\text{exp}})$  which consists of a (long) implicit input  $x_{\text{imp}}$  and a (short) explicit input  $x_{\text{exp}}$ , and outputs a bit  $b = \mathcal{R}(x) \in \{0, 1\}$  and a proof  $\pi$ .

$\mathcal{V}(\text{crs}, \text{hk}, \mathbf{v}, x_{\text{exp}}, b, \pi) \rightarrow \{0, 1\}$ . This is a deterministic verifier that takes as input a  $\text{crs}$ , an SEH hash key  $\text{hk}$ , a digest  $\mathbf{v}$  of the (long) implicit input, a (short) explicit input  $x_{\text{exp}}$ , a bit  $b \in \{0, 1\}$ , and a proof  $\pi$ , and outputs 1 (accept) or 0 (reject).

---

<sup>4</sup>We note that we require that  $\text{Gen}$  runs in time poly-logarithmically in  $n$ . This may not be an inherent requirement for a rate-1 BARG. We add this requirement since we can achieve it, and we use it in Section 7 where we construct a multi-hop seBARG scheme and an aggregate signature scheme.

**Remark 3.5.** Sometimes the goal of the prover  $\mathcal{P}$  is to prove that the RAM computation is accepting (i.e., outputs 1). In this case we omit the bit  $b$  (which is set to 1), and simply think of  $\mathcal{V}$  as taking as input the tuple  $(\text{crs}, \text{hk}, \mathbf{v}, x_{\text{exp}}, \pi)$ . This will indeed be the case in Sections 4 and 5.

**Definition 3.7.** A flexible RAM SNARG  $= (\text{Gen}, \text{Digest}, \mathcal{P}, \mathcal{V})$  associated with a somewhere extractable hash family (Section 3.1)  $\text{SEH} = (\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify})$ , is required to satisfy the following properties:

**Efficiency.** The length of the RAM proof  $\pi$  is at most  $\text{poly}(\lambda, T_V)$  where  $T_V$  is a bound on the run-time of  $\text{SEH.Verify}$ .

In particular, if the hash key can be partitioned into  $\text{hk} = (\text{hk}_\ell, \text{hk}_s)$  and the hash value can be partitioned into  $\mathbf{v} = (\mathbf{v}_\ell, \mathbf{v}_s)$ , and  $\text{SEH.Verify}$  takes as input only  $(\text{hk}_s, \mathbf{v}_s)$ , together with  $(j, b, \rho)$ , then  $|\pi| \leq \text{poly}(|\text{hk}_s|, |\mathbf{v}_s|, |\rho|, \lambda)$ , and thus the run-time of  $\mathcal{V}$  is at most  $\text{poly}(|\text{hk}_s|, |\mathbf{v}_s|, |\rho|, \lambda)$ .

**Remark 3.6.** Jumping ahead, we note that if the underlying SEH hash family is a fully local SEH family (defined in Section 4) then  $|\text{hk}_s|, |\mathbf{v}_s|, |\rho| \leq \text{poly}(\lambda)$  in which case the length of  $\pi$  and the run-time of  $\mathcal{V}$  is at most  $\text{poly}(\lambda)$ .

**Completeness.** For any  $\lambda, N \in \mathbb{N}$  such that  $N \leq T(N) \leq 2^\lambda$  and any  $x = (x_{\text{imp}}, x_{\text{exp}}) \in \{0, 1\}^N$  such that  $\mathcal{R}(x)$  halts within  $T$  time steps, and any  $I \in [|x_{\text{imp}}|]$  we have that

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, \text{hk}, \mathbf{v}, x_{\text{exp}}, b, \pi) = 1 \\ \wedge b = \mathcal{R}(x) \end{array} : \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda), \\ (\text{hk}, \text{td}) \leftarrow \text{SEH.Gen}(1^\lambda, |x_{\text{imp}}|, I), \\ (b, \pi) = \mathcal{P}(\text{crs}, \text{hk}, x), \\ \mathbf{v} = \text{Digest}(\text{hk}, x_{\text{imp}}) \end{array} \right] = 1.$$

**Partial-input soundness.** For any poly-size adversary  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  and any polynomial  $T = T(\lambda)$  there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, \text{hk}, \mathbf{v}, x_{\text{exp}}, b^*, \pi) = 1 \\ \wedge \mathcal{R}(x_{\text{imp}}, x_{\text{exp}}, T) = 1 - b^* \\ \text{and does not read any} \\ \text{location in } [N] \setminus I \end{array} : \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda), \\ (1^N, I) = \mathcal{A}_1(\text{crs}), \\ (\text{hk}, \text{td}) \leftarrow \text{SEH.Gen}(1^\lambda, N, I), \\ (\mathbf{v}, x_{\text{exp}}, b^*, \pi) = \mathcal{A}_2(\text{crs}, \text{hk}), \\ (b_j)_{j \in I} = \text{SEH.Extract}(\text{td}, \mathbf{v}, I), \\ \text{define } x_{\text{imp}} \in \{0, 1\}^N : \\ \forall j \in I, (x_{\text{imp}})_j = b_j; \forall j \in [N] \setminus I, (x_{\text{imp}})_j = 0 \end{array} \right] \leq \text{negl}(\lambda).$$

**Theorem 3.8** ([KLVW22]). Assuming the hardness of the LWE problem, there exists a flexible RAM SNARG.

### 3.4 Homomorphic Encryption with Ciphertext Compression

Below we recall the notion of a homomorphic encryption scheme, and define ciphertext compression algorithms. These ciphertext compression algorithms are inspired by the rate-1 homomorphic encryption scheme of Brakerski et al. [BDGM19], and are essential in our construction of a rate-1 fully-local somewhere extractable hash family (defined in section 4). Our compressed ciphertexts have desired locality properties that are essential to our fully-local somewhere extractable hash family.

**Syntax.** A homomorphic encryption scheme HE for a circuit class  $\mathcal{C}$  and message space  $\{0, 1\}$  consists of the following PPT algorithms:

$\text{Gen}(1^\lambda) \rightarrow (\text{pk}, \text{sk})$ . This is a setup algorithm that takes as input the security parameter  $\lambda$ , and outputs a public-secret key pair  $(\text{pk}, \text{sk})$ .

$\text{Enc}(\text{pk}, b) \rightarrow \text{ct}$ . The encryption algorithm takes as input a public key  $\text{pk}$ , message bit  $b \in \{0, 1\}$ , and outputs a ciphertext  $\text{ct}$ .

$\text{Eval}(\text{pk}, C, (\text{ct}_1, \dots, \text{ct}_n)) \rightarrow (\text{ct}'_1, \dots, \text{ct}'_\ell)$ . The evaluation algorithm is a deterministic algorithm that takes as input a public key  $\text{pk}$ , description of a circuit  $C \in \mathcal{C}$ , and  $n$  ciphertexts  $\text{ct}_i$  for  $i \in [n]$ , where  $n$  is the input length of the circuit  $C$ . It outputs a sequence of evaluated ciphertexts  $(\text{ct}'_1, \dots, \text{ct}'_\ell)$ , where  $\ell$  is the output length of the circuit  $C$ .

$\text{Dec}(\text{sk}, \text{ct}) \rightarrow b$ . The decryption algorithm is a deterministic algorithm that takes as input a secret key  $\text{sk}$ , a (possibly evaluated) ciphertext  $\text{ct}$ , and outputs a message bit  $b$ .

(Throughout the sequel, we naturally define the decryption algorithm to take as input a sequence of ciphertexts, and output a sequence of message bits as output.)

**Definition 3.9** (Correctness and Compactness of HE). *The encryption scheme  $\text{HE} = (\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  is said to be correct if for any security parameter  $\lambda \in \mathbb{N}$ , any circuit  $C \in \mathcal{C}$  with input and output lengths  $n$  and  $\ell$  (respectively), and any sequence of  $n$  messages  $b_1, \dots, b_n \in \{0, 1\}$ ,*

$$\Pr[\text{Dec}(\text{sk}, \text{Eval}(\text{pk}, C, (\text{ct}_1, \dots, \text{ct}_n))) = C(b_1, \dots, b_n)] = 1$$

where the probability is over  $(\text{pk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda)$  and  $\text{ct}_i \leftarrow \text{Enc}(\text{pk}, b_i)$  for  $i \in [n]$ .

The encryption scheme HE is said to be compact if the bit-length of the evaluated ciphertext is at most  $\text{poly}(\lambda, \ell)$ , i.e. its size does not depend on circuit size.

**Compressing ciphertexts.** In our construction of a rate-1 fully-local SEH (in Section 4), we use a homomorphic encryption scheme that has a “compression” algorithm which takes a set of  $\ell$  evaluated ciphertexts (each encrypting a single bit) and compresses them into a single rate-1 ciphertext encrypting these  $\ell$  bits. Moreover, we require this “compression” algorithm to have a specific form which is needed in order to construct the rate-1 fully-local SEH scheme.

The compression algorithm consists of three parts:  $\text{Compress}_1, \text{LinEval}, \text{Compress}_2$ , which we elaborate on below.<sup>5</sup> We note that in order to compress  $\ell$  ciphertexts we need to generate new “compression” keys (that depend on  $\ell$ ), via an algorithm  $\text{CompGen}$  and we need a new decryption algorithm called  $\text{CompDec}$ . We require this decryption algorithm to be local (as defined below).

$\text{CompGen}(\text{pk}, \text{sk}, 1^\ell) \rightarrow (\text{pk}_c, \text{sk}_c)$ . The PPT compression key generator algorithm takes as input a public-secret key pair  $(\text{pk}, \text{sk})$ , and the compression parameter  $\ell$ , and it outputs a new public-secret key pair  $(\text{pk}_c, \text{sk}_c)$  that enables ciphertext compression.

(Here  $\ell$  should be regarded as the maximum number of one-bit ciphertexts that can be compressed to a rate-1 encryption.)

---

<sup>5</sup>We note that the properties we specify below are quite specific. We could have specified more general properties but we chose simplicity over generality.

$\text{Compress}_1(\text{pk}_c, \text{ct}, \ell, i) \rightarrow \text{ct}_{c_1}$ . The first part of the compression algorithm is a deterministic poly-time algorithm that takes as input a compression public key  $\text{pk}_c$ , a (non-compressed) HE ciphertext  $\text{ct}$  encrypting a single bit  $b$ , a length value  $\ell \in \mathbb{N}$ , and an index  $i \in [\ell]$ . It outputs a “processed” ciphertext  $\text{ct}_{c_1}$ , which encrypts the  $\ell$ -bit vector  $b \cdot e_i \in \{0, 1\}^\ell$  where  $e_i \in \{0, 1\}^\ell$  is 1 on its  $i$ ’th coordinate and zero on all other coordinates.

This processed ciphertext has the form

$$\text{ct}_{c_1} = (\text{sub-ct}_0, \text{sub-ct}_1, \dots, \text{sub-ct}_\ell),$$

where  $\text{sub-ct}_0$  is a “preamble” consisting of  $\text{poly}(\lambda)$  field elements in a finite field  $\mathbb{F}_q$  of size  $q \leq 2^\lambda$ , and each  $\text{sub-ct}_i$  for  $i \in [\ell]$  consists of a single element in  $\mathbb{F}_q$ .<sup>6</sup>

$\text{LinEval}(\text{pk}, \text{ct}_{c_1}^{(1)}, \dots, \text{ct}_{c_1}^{(\ell)}) \rightarrow \text{ct}'_{c_1}$ . This linear evaluation algorithm is a deterministic poly-time algorithm that takes as input a public key  $\text{pk}$ , and  $\ell$  processed ciphertexts (each encrypting an  $\ell$ -bit vector), and outputs a single processed ciphertext (encrypting an  $\ell$ -bit vector).

This algorithm simply adds the  $\ell$  ciphertexts  $\text{ct}_{c_1}^{(1)}, \dots, \text{ct}_{c_1}^{(\ell)}$  coordinate-wise over  $\mathbb{F}_q$ . Importantly, if each  $\text{ct}_{c_1}^{(i)}$  encrypts the vector  $b_i \cdot e_i \in \{0, 1\}^\ell$  then the output ciphertext  $\text{ct}'_{c_1}$  encrypts the vector  $(b_1, \dots, b_\ell)$ .

$\text{Compress}_2(\text{ct}_{c_1}) \rightarrow \text{ct}_c$ . The final compression algorithm is a deterministic poly-time algorithm that takes as input a processed ciphertext  $\text{ct}_{c_1}$  and outputs a fully compressed ciphertext  $\text{ct}_c$ .

Importantly, this algorithm can be executed locally in the sense that one can run it on each sub-ciphertext separately. Namely,  $\text{Compress}_2(\text{ct}_{c_1})$  can be computed as follows:

1. Parse  $\text{ct}_{c_1} = (\text{sub-ct}_0, \text{sub-ct}_1, \dots, \text{sub-ct}_\ell)$ .
2. For every  $j \in [\ell]$ , compute  $\text{sub-ct}_{c,j} = \text{Compress}_2(\text{sub-ct}_j) \in \{0, 1\}$ .  
(For the sake of simplicity, we are overloading the notation of  $\text{Compress}_2$ .)
3. Output  $\text{ct}_c = (\text{sub-ct}_0, \text{sub-ct}_{c,1}, \dots, \text{sub-ct}_{c,\ell})$ .

$\text{CompDec}(\text{sk}_c, \text{ct}_c) \rightarrow (b_1, \dots, b_\ell)$ . This is a deterministic poly-time decryption algorithm that decrypts compressed ciphertexts. It takes as input a compression secret key  $\text{sk}_c$  and a compressed ciphertext  $\text{ct}_c$  and it outputs its decryption  $(b_1, \dots, b_\ell)$ .

Importantly, this algorithm is local in the following sense: Parsing  $\text{ct}_c = (\text{sub-ct}_0, \text{sub-ct}_{c,1}, \dots, \text{sub-ct}_{c,\ell})$  (and overloading notation),

$$\text{CompDec}(\text{sk}_c, \text{ct}_c) = (\text{CompDec}(\text{sk}_c, \text{sub-ct}_0, \text{sub-ct}_1), \dots, \text{CompDec}(\text{sk}_c, \text{sub-ct}_0, \text{sub-ct}_\ell)).$$

**Definition 3.10.** *[Correctness of Compressed Encryption Scheme] A homomorphic encryption scheme  $\text{HE} = (\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  for a circuit class  $\mathcal{C}$ , with compression algorithms*

$$(\text{CompGen}, \text{Compress}_1, \text{LinEval}, \text{Compress}_2, \text{CompDec})$$

*as defined above, is said to be correct and compact if the encryption scheme is correct and compact (as per definition 3.9), and there exists a negligible function  $\text{negl}$  such that for any security parameter*

---

<sup>6</sup>We note that  $\mathbb{F}_q$  is the underlying field used in this encryption scheme.

$\lambda \in \mathbb{N}$ , parameters  $\ell, n \leq 2^\lambda$ , key pair  $(\text{pk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda)$ , compressed key pair  $(\text{pk}_c, \text{sk}_c) \leftarrow \text{CompGen}(\text{pk}, \text{sk}, 1^\ell)$ , any circuit  $C \in \mathcal{C}$  with input and output lengths  $n$  and  $\ell$  respectively, any sequence of  $n$  bit messages  $b_1, \dots, b_n \in \{0, 1\}$ , ciphertexts  $\text{ct}_i \leftarrow \text{Enc}(\text{pk}, b_i)$  for  $i \in [n]$ , evaluated ciphertext

$$(\text{ct}'_1, \dots, \text{ct}'_\ell) = \mathbf{ct}' \leftarrow \text{Eval}(\text{pk}, C, (\text{ct}_1, \dots, \text{ct}_n)),$$

partially compressed ciphertexts

$$(\text{sub-ct}'_0^{(i)}, \text{sub-ct}'_1^{(i)}, \dots, \text{sub-ct}'_\ell^{(i)}) = \text{ct}'_{c_1}{}^{(i)} \leftarrow \text{Compress}_1(\text{pk}_c, \text{ct}'_i, \ell, i)$$

for  $i \in [\ell]$ , letting

$$\text{ct}'_{c_1} = \text{LinEval}(\text{pk}, \text{ct}'_{c_1}{}^{(1)}, \dots, \text{ct}'_{c_1}{}^{(\ell)}),$$

and  $\text{ct}_c = \text{Compress}_2(\text{ct}'_{c_1})$ , it holds that

$$\Pr[\text{CompDec}(\text{sk}_c, \text{ct}_c) = C(b_1, \dots, b_n)] \geq 1 - \text{negl}(\lambda).$$

where the probability is over the sampling of all the ciphertexts above (including the ciphertexts  $\text{ct}_1, \dots, \text{ct}_n$ ).

**Security.** For security of our encryption scheme with compression algorithms, we consider an extended notion of semantic security, where the attacker gets both the regular and compression public keys, and it still can not distinguish between two honestly computed ciphertexts.

**Definition 3.11** (Semantic Security). *A homomorphic encryption scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  with compression algorithms  $(\text{CompGen}, \text{Compress}_1, \text{LinEval}, \text{Compress}_2, \text{LocDec}_2)$  is said to be secure if for every poly-size adversary  $\mathcal{A}$  there exists a negligible function  $\text{negl}(\cdot)$  such that for all  $\lambda \in \mathbb{N}$ , the following holds:*

$$\Pr \left[ \begin{array}{l} \mathcal{A}(\text{pk}, \text{pk}_c, \text{ct}) = b : \\ \begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda), 1^\ell \leftarrow \mathcal{A}(\text{pk}) \\ (\text{pk}_c, \text{sk}_c) \leftarrow \text{CompGen}(\text{pk}, \text{sk}, 1^\ell) \\ b \leftarrow \{0, 1\}, \text{ct} \leftarrow \text{Enc}(\text{pk}, b) \end{array} \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

Later in appendix A, we construct such a homomorphic encryption scheme with compression algorithms by relying on the rate-1 FHE construction by Brakerski et al. [BDGM19]. Below we state the main theorem that we prove in appendix A.

**Theorem 3.12.** *Assuming the hardness of the LWE problem, there exists a homomorphic encryption scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  for all circuits of depth  $\text{poly}(\lambda)$  (where the public key grows with  $\text{poly}(\lambda)$ ), with compression algorithms  $(\text{CompGen}, \text{Compress}_1, \text{LinEval}, \text{Compress}_2, \text{CompDec})$  satisfying all the properties defined above, including correctness, compactness, and security (see Definitions 3.9 to 3.11).*

## 4 Rate-1 Fully-Local SEH (flSEH) Families

### 4.1 Definition

**Syntax.** A rate-1 fully-local somewhere extractable hash family consists of the following polynomial time algorithms:

$\text{Gen}(1^\lambda, N, I) \rightarrow (\text{crs}_\ell, \text{crs}_s, \text{td})$ . This is a probabilistic poly-time setup algorithm that takes as input security parameter  $1^\lambda$  in unary, message length  $N$ , and subset  $I \subset [N]$ . It runs in time  $\text{poly}(\lambda, |I|, \log N)$  and outputs a long crs  $\text{crs}_\ell \in \{0, 1\}^{|I| \cdot \text{poly}(\lambda)}$ , a short crs  $\text{crs}_s \in \{0, 1\}^{\text{poly}(\lambda)}$ , and a trapdoor  $\text{td}$ .

Moreover, if the set  $I$  consists of consecutive indices  $I = \{i^* + 1, \dots, i^* + m\}$  then  $\text{Gen}$  runs in time  $\text{poly}(\lambda, \log N, \log m)$  and outputs  $(\text{crs}, \text{td})$  both of size  $\text{poly}(\lambda, \log N, \log m)$ .<sup>7</sup>

$\text{Hash}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}) \rightarrow (\mathbf{v}, \text{rt})$ . This is a deterministic poly-time hash algorithm that takes as input (long)  $\text{crs}_\ell$ , (short) crs  $\text{crs}_s$ , and message  $\mathbf{x} \in \{0, 1\}^N$ , and outputs a long hash value  $\mathbf{v} \in \{0, 1\}^{|I| + \text{poly}(\lambda)}$  and a short digest  $\text{rt} \in \{0, 1\}^{\text{poly}(\lambda)}$ .

**Remark 4.1.** We note that we can always assume w.l.o.g. that  $\text{crs}_\ell$  contains  $\text{crs}_s$ , in which case  $\text{Hash}$  can take as input only  $\text{crs}_\ell$ . However, in our construction (in Section 4.2)  $\text{crs}_\ell$  does not include  $\text{crs}_s$  and hence we give  $\text{Hash}$  both  $\text{crs}_\ell$  and  $\text{crs}_s$ .

$\text{Validate}(\text{crs}_s, \mathbf{v}, \text{rt}) \rightarrow 0/1$ . This is a deterministic poly-time validation algorithm that takes as input (short)  $\text{crs}_s$ , hash value  $\mathbf{v}$ , and (short) digest  $\text{rt}$ , and outputs 1 (accept) or 0 (reject).

$\text{Open}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}, i) \rightarrow (x_i, \rho)$ . This is a deterministic poly-time opening algorithm that takes as input (long)  $\text{crs}_\ell$ , (short) crs  $\text{crs}_s$ , message  $\mathbf{x} \in \{0, 1\}^N$ , and index  $i \in [N]$ , and outputs a bit  $x_i \in \{0, 1\}$  and a local opening  $\rho$  of length  $\text{poly}(\lambda, \log N)$ .

$\text{Verify}(\text{crs}_s, \text{rt}, i, b, \rho) \rightarrow 0/1$ . This is a deterministic poly-time verification algorithm that takes as input (short)  $\text{crs}_s$ , (short) digest  $\text{rt}$  (produced by  $\text{Hash}$ ), index  $i \in [N]$ , bit  $b$ , and local opening  $\rho$ , and outputs 1 (accept) or 0 (reject).<sup>8</sup>

$\text{Extract}(\text{td}, \mathbf{v}) \rightarrow u$ . This is a deterministic poly-time extraction algorithm that takes as input trapdoor  $\text{td}$ , and hash value  $\mathbf{v}$ , and outputs a string  $u \in \{0, 1\}^{|I|}$ .

A rate-1 fISEH family is required to satisfy similar properties to those of a SEH family adapted to the fISEH setting. The adapted properties are described formally below.

**Definition 4.1** (rate-1 fISEH). *A rate-1 fISEH family is required to have the following properties:*

**Index hiding.** *For any poly-size adversary  $\mathcal{A}$ , any polynomial  $N = N(\lambda)$ , and any subsets  $I_0, I_1 \in [N]$  such that  $|I_0| = |I_1|$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,*

$$\Pr \left[ \mathcal{A}(\text{crs}_\ell, \text{crs}_s) = b : \begin{array}{l} b \leftarrow \{0, 1\} \\ (\text{crs}_\ell, \text{crs}_s, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

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

$$\Pr \left[ \begin{array}{l} b = x_i \\ \wedge \text{Verify}(\text{crs}_s, \text{rt}, i, b, \rho) = 1 \\ \wedge \text{Validate}(\text{crs}_s, \mathbf{v}, \text{rt}) = 1 \end{array} : \begin{array}{l} (\text{crs}_\ell, \text{crs}_s, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (\mathbf{v}, \text{rt}) = \text{Hash}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}), \\ (b, \rho) = \text{Open}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}, i), \end{array} \right] = 1.$$

<sup>7</sup>In this case, there is no reason to distinguish between  $\text{crs}_\ell$  and  $\text{crs}_s$  since  $\text{crs}$  is short.

<sup>8</sup>Note that the run-time of  $\text{Verify}$  is  $\text{poly}(\lambda)$  assuming  $N \leq 2^\lambda$ .



**Somewhere statistically binding w.r.t. opening.** For any  $\lambda \in \mathbb{N}$ , any  $N \leq 2^\lambda$ , any subset  $I \subseteq [N]$ , any index  $i \in I$ , and any (all powerful) adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{Verify}(\text{crs}_s, \text{rt}, i, b, \rho) = 1 \\ \wedge \text{Validate}(\text{crs}_s, \mathbf{v}, \text{rt}) = 1 \\ \wedge b \neq b_i \end{array} : \begin{array}{l} (\text{crs}_\ell, \text{crs}_s, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (\mathbf{v}, \text{rt}, b, \rho) \leftarrow \mathcal{A}(\text{crs}_\ell, \text{crs}_s), \\ (b_j)_{j \in I} = \text{Extract}(\text{td}, \mathbf{v}) \end{array} \right] \leq \text{negl}(\lambda).$$

**Extraction correctness.** For any  $\lambda \in \mathbb{N}$ , any  $N \leq 2^\lambda$ , any subset  $I \subseteq [N]$ , and any  $\mathbf{x} \in \{0, 1\}^N$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ x_I \neq \text{Extract}(\text{td}, \mathbf{v}) : \begin{array}{l} (\text{crs}_\ell, \text{crs}_s, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I) \\ \mathbf{v} \leftarrow \text{Hash}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}) \end{array} \right] \leq \text{negl}(\lambda).$$

**Remark 4.2.** One can think of a fISEH as a SEH with a *offline/online opening verification*, where in the offline phase `Validate` is executed and in the online phase `Verify` is executed.

## 4.2 Construction

Our construction of a fISEH family uses the following ingredients:

1. A rate-1 linearly homomorphic or levelled fully homomorphic encryption scheme

$$(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$$

with compression algorithms

$$(\text{CompGen}, \text{Compress}_1, \text{LinEval}, \text{Compress}_2, \text{CompDec})$$

(see Section 3.4). Recall that each ciphertext is a vector over a finite field, denoted by  $\mathbb{F}_q$ .

2. A somewhere extractable hash family

$$\text{SEH} = (\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify}, \text{SEH.Extract})$$

constructed w.r.t. the rate-1 encryption scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  from Item 1 (see Section 3.1 and Lemma 3.2).

3. Two flexible RAM SNARG schemes with partial input soundness (see Section 3.3):

$$(\text{RAM}_0.\text{Gen}, \text{RAM}_0.\text{Digest}, \text{RAM}_0.\mathcal{P}, \text{RAM}_0.\mathcal{V})$$

and

$$(\text{RAM}_1.\text{Gen}, \text{RAM}_1.\text{Digest}, \text{RAM}_1.\mathcal{P}, \text{RAM}_1.\mathcal{V}),$$

where the first is w.r.t. the RAM machine  $\mathcal{R}_0$  that runs in time  $T_0$  and the second is w.r.t. the RAM machine  $\mathcal{R}_1$  that runs in time  $T_1$ .

- (a) The RAM machine  $\mathcal{R}_0$  is associated with the encryption scheme from Item 1. It takes as input a sequence of ciphertexts  $(\text{ct}_1, \dots, \text{ct}_m)$  each encrypting a single bit, the corresponding public key  $\text{pk}$ , indices  $j \in [m]$  and  $k \in [m']$ , where  $m'$  denotes the number of field elements in the vector  $\text{Compress}_1(\text{pk}, \text{ct}_j, m, j)$ , and a field element  $c \in \mathbb{F}_q$ , and outputs 1 if and only if  $c$  is the  $k$ 'th element of  $\text{Compress}_1(\text{pk}_c, \text{ct}_j, m, j)$ .

Recall that  $m' = m + \text{poly}(\lambda)$ . We let

$$\Delta(\lambda) \triangleq m' - m = \text{poly}(\lambda) \quad (1)$$

- (b) The RAM machine  $\mathcal{R}_1$  is specified in Item 6b of the Hash algorithm construction below.

4. Two somewhere extractable BARG schemes (see Section 3.2)

$$(\text{seBARG}_0.\text{Gen}, \text{seBARG}_0.\mathcal{P}, \text{seBARG}_0.\mathcal{V})$$

and

$$(\text{seBARG}_1.\text{Gen}, \text{seBARG}_1.\mathcal{P}, \text{seBARG}_1.\mathcal{V})$$

where the first is for the NP language  $\mathcal{L}_0$ , specified in Item 9 of the Hash algorithm construction below, and the second is for the NP language  $\mathcal{L}_1$ , specified in Item 5 of the Open algorithm construction below.

We are now ready to define our fISEH family.

$\text{Gen}(1^\lambda, N, I) \rightarrow (\text{crs}_\ell, \text{crs}_s, \text{td})$ . This poly-time algorithm does the following:

In what follows we let  $m = |I|$  and we assume without loss of generality that  $m$  is a power of 2, and for every  $t \in [0, \log m]$  we denote by  $m_t = m/2^t$ .

1. Generate  $(\text{hk}', \text{td}') \leftarrow \text{SEH.Gen}(1^\lambda, N, I)$ .
2. Parse  $\text{hk}' = (\text{pk}, \text{hk}'_1, \dots, \text{hk}'_m)$ .  
Recall that  $\text{td}' = \text{sk}$  where  $\text{sk}$  is the secret key corresponding to  $\text{pk}$ , and  $n \triangleq |\text{hk}'_j| \leq \text{poly}(\lambda)$  for every  $j \in [m]$  (see Lemma 3.2). In addition, recall that each hash value corresponding to  $\text{hk}'$  consists of  $m = |I|$  ciphertexts w.r.t.  $\text{pk}$ , each encrypting a single bit, where the  $j$ 'th ciphertext is a function of  $\text{pk}$  and  $\text{hk}'_j$ . By padding, we can assume w.l.o.g. that the size of each of these ciphertexts is also  $n$ .
3. Generate  $(\text{pk}_c, \text{sk}_c) \leftarrow \text{CompGen}(\text{pk}, \text{sk}, 1^m)$ .
4. Set arbitrarily  $\alpha, \beta \in [m]$ . For example, set  $\alpha = \beta = 1$ .
5. Generate  $(\text{hk}, \text{td}) \leftarrow \text{SEH.Gen}(1^\lambda, m \cdot n, I_\beta)$ , where  $I_\beta = \{(\beta - 1) \cdot n + 1, \dots, \beta \cdot n\}$ .
6. Let  $\text{h}' = \text{SEH.Hash}(\text{hk}, (\text{hk}'_1, \dots, \text{hk}'_m))$ .
7. For every  $t \in [0, \log m]$ , generate  $(\text{hk}_t, \text{td}_t) \leftarrow \text{SEH.Gen}(1^\lambda, m' \cdot m_t \cdot n_t, J_{t,\alpha,\beta})$ , where the value  $n_t$  and the set of coordinates  $J_{t,\alpha,\beta} \subseteq [m' \cdot m_t \cdot n_t]$  of size  $\text{poly}(\lambda)$  are specified in Item 6a of the Hash algorithm construction below.
8. Generate  $\text{RAM}_0.\text{crs} \leftarrow \text{RAM}_0.\text{Gen}(1^\lambda, T_0)$ .
9. Generate  $\text{RAM}_1.\text{crs} \leftarrow \text{RAM}_1.\text{Gen}(1^\lambda, T_1)$ .

10. Generate  $(\text{seBARG}_0.\text{crs}, \text{seBARG}_0.\text{td}) \leftarrow \text{seBARG}_0.\text{Gen}(1^\lambda, m', n', S_\alpha)$ , where recall that  $m' = \Delta(\lambda) + m$  is the number of field elements in the vector  $\text{Compress}_1(\text{pk}, \text{ct}_j, m, j)$ ,  $n' \leq \text{poly}(\lambda)$  is specified in Item 9 of the Hash algorithm construction below, and  $S_\alpha = \{1, \dots, \Delta(\lambda)\} \cup \{\Delta(\lambda) + \alpha\} \subseteq [m']$ .
11. Generate  $(\text{seBARG}_1.\text{crs}, \text{seBARG}_1.\text{td}) \leftarrow \text{seBARG}_1.\text{Gen}(1^\lambda, m, n'', \alpha)$ , where  $n'' \leq \text{poly}(\lambda)$  is specified in Item 5 of the Open algorithm construction below.
12. Let  $\text{crs}_\ell = \text{hk}'$  and

$$\text{crs}_s = \left( \text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs} \right).$$

13. Output  $(\text{crs}_\ell, \text{crs}_s, \text{td}^*)$ , where  $\text{td}^* = \text{sk}_c$ .

**Remark 4.3.** Note that if  $I = \{i^* + 1, \dots, i^* + m\}$  then  $\text{hk}'$  can be computed efficiently from  $\text{pk}$  and an encryption of  $i^*$ , so  $\text{Gen}$  runs in time  $\text{poly}(\lambda, \log N)$  (see Lemma 3.2).

$\text{Hash}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}) \rightarrow (\mathbf{v}, \text{rt})$ . This poly-time algorithm does the following:

1. Parse  $\text{crs}_\ell = \text{hk}' = (\text{pk}, \text{hk}'_1, \dots, \text{hk}'_m)$  and

$$\text{crs}_s = \left( \text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs} \right).$$

2. Compute

$$(\text{ct}_1, \dots, \text{ct}_m) = \text{SEH.Hash}(\text{hk}', \mathbf{x}).$$

Denoting by  $I = \{i_1, \dots, i_m\} \subseteq [N]$ , note that  $\text{ct}_j$  is a ciphertext encrypting the bit  $x_{i_j}$  w.r.t.  $\text{pk}$  (see Lemma 3.2).

3. Compute

$$\mathbf{h} = \text{SEH.Hash}(\text{hk}, (\text{ct}_1, \dots, \text{ct}_m)).$$

Note that  $|\mathbf{h}| \leq \text{poly}(\lambda)$ .

4. Compute  $\mathbf{v}$  which is the compressed ciphertext corresponding to  $(\text{ct}_1, \dots, \text{ct}_m)$  and  $\text{pk}_c$ , as follows:

- (a) For every  $j \in [m]$  compute

$$(\text{ct}_j)_{c_1} = \text{Compress}_1(\text{pk}_c, \text{ct}_j, m, j).$$

Note that  $(\text{ct}_j)_{c_1}$  is a ciphertext encrypting the vector  $x_{i_j} \cdot e_j \in \{0, 1\}^m$ .

- (b) Compute

$$\text{ct}_{c_1} = \text{LinEval}(\text{pk}, (\text{ct}_1)_{c_1}, \dots, (\text{ct}_m)_{c_1}).$$

- (c) Compute

$$\mathbf{v} = \text{Compress}_2(\text{ct}_{c_1}).$$

Note that  $\mathbf{v} = (\mathbf{v}_1, \dots, \mathbf{v}_{m'})$  is a ciphertext encrypting the string  $x_I \in \{0, 1\}^m$ , where  $\mathbf{v}_k \in \mathbb{F}_q$  for every  $k \in [\Delta(\lambda)]$ , and  $\mathbf{v}_k \in \{0, 1\}$  for every  $k \in [\Delta(\lambda) + 1, m']$ .<sup>9</sup>

In Item 5-Item 10 below we compute a proof that  $\mathbf{h}$  and  $\mathbf{v}$  are consistent.

---

<sup>9</sup>Using our notation in Section 3.4,  $\text{sub-ct}_0 = (\mathbf{v}_1, \dots, \mathbf{v}_{\Delta(\lambda)}) \in \mathbb{F}_q^{\Delta(\lambda)}$  and  $\text{sub-ct}_{e,j} = \mathbf{v}_{\Delta(\lambda)+j} \in \{0, 1\}$  for every  $j \in [m]$ .

5. For every  $j \in [m]$  and  $k \in [m']$  let

$$c_{j,k}^{(0)} = k\text{-th element of } (\text{ct}_j)_{\mathbf{c}_1}, \text{ and}$$

$$\pi_{j,k}^{(0)} = \text{RAM}_0.\mathcal{P} \left( \text{RAM}_0.\text{crs}, \text{hk}, (\text{ct}_1, \dots, \text{ct}_m), (\text{pk}, j, k, c_{j,k}^{(0)}) \right).$$

Note that  $\pi_{j,k}^{(0)}$  is a RAM proof that  $c_{j,k}^{(0)}$  is the  $k$ 'th element of  $\text{Compress}_1(\text{pk}_{\mathbf{c}}, \text{ct}_j, m, j)$ .

6. For  $t = 0, \dots, \log m$ :

(a) Let

$$X^{(t)} = \left\{ c_{j,k}^{(t)}, \pi_{j,k}^{(t)} \right\}_{j \in [m_t], k \in [m']} \quad (2)$$

and compute

$$\text{hk}_t = \text{SEH.Hash} \left( \text{hk}_t, X^{(t)} \right).$$

where  $\text{hk}_t$  is defined to hash messages of length  $m' \cdot m_t \cdot n_t$  and be statistically binding on the coordinates in  $J_{t,\alpha,\beta}$ , where  $n_t$  and  $J_{t,\alpha,\beta}$  are defined as follows:

$$n_t = \log q + \left\lceil \pi_{j,k}^{(t)} \right\rceil$$

where  $\left\lceil \pi_{j,k}^{(t)} \right\rceil$  is the number of bits in the proof  $\pi_{j,k}^{(t)}$ , and  $J_{t,\alpha,\beta}$  is defined by first letting

$$J'_{t,\beta} = \left\{ \lceil \beta/2^t \rceil - 1, \lceil \beta/2^t \rceil, \lceil \beta/2^t \rceil + 1 \right\} \cap [m_t]$$

and letting  $J_{t,\alpha,\beta}$  be the set of coordinates corresponding to  $(c_{j,k}^{(t)}, \pi_{j,k}^{(t)})$  for  $j \in J'_{t,\beta}$  and  $k \in S_\alpha$ .<sup>10</sup> Note that  $|J_{t,\alpha,\beta}| \leq \text{poly}(\lambda)$ .

(b) If  $t < \log m$ : For every  $j \in [m_{t+1}]$  and  $k \in [m']$ , compute

$$c_{j,k}^{(t+1)} = c_{2j-1,k}^{(t)} + c_{2j,k}^{(t)},$$

where addition is over the finite field  $\mathbb{F}_q$  used in the underlying encryption scheme (see Definition 3.10), and compute

$$\pi_{j,k}^{(t+1)} = \text{RAM}_1.\mathcal{P} \left( \text{RAM}_1.\text{crs}, \text{hk}_t, X^{(t)}, \left( \text{hk}, \text{h}, \{\text{hk}_\ell, \text{h}_\ell\}_{\ell=0}^{t-1}, \text{pk}, j, k, c_{j,k}^{(t+1)} \right) \right).$$

$\text{RAM}_1$  is defined w.r.t. the RAM machine  $\mathcal{R}_1$  that takes as input  $X^{(t)}$  as in Equation (2), hash keys  $(\text{hk}, \text{hk}_0, \dots, \text{hk}_{t-1})$ , hash values  $(\text{h}, \text{h}_0, \text{h}_1, \dots, \text{h}_{t-1})$ , the public key  $\text{pk}$  defined by  $\text{hk}'$ , indices  $j \in [m_{t+1}]$  and  $k \in [m']$ , and field element  $c_{j,k}^{(t+1)}$ , and outputs 1 if and only if the following conditions are satisfied:

- i.  $c_{j,k}^{(t+1)} = c_{2j-1,k}^{(t)} + c_{2j,k}^{(t)}$  over  $\mathbb{F}_q$ .
- ii. For every  $b \in \{0, 1\}$ ,  $\pi_{2j-b,k}^{(t)}$  is a valid proof. Namely, if  $t = 0$ , then

$$\text{RAM}_0.\mathcal{V} \left( \text{RAM}_0.\text{crs}, \text{hk}, \text{h}, \left( \text{pk}, 2j - b, k, c_{2j-b,k}^{(t)} \right), \pi_{2j-b,k}^{(t)} \right) = 1$$

and if  $t \geq 1$ , then

$$\text{RAM}_1.\mathcal{V} \left( \text{RAM}_1.\text{crs}, \text{hk}_{t-1}, \text{h}_{t-1}, \left( \text{hk}, \text{h}, \{\text{hk}_\ell, \text{h}_\ell\}_{\ell=0}^{t-2}, \text{pk}, 2j - b, k, c_{2j-b,k}^{(t)} \right), \pi_{2j-b,k}^{(t)} \right) = 1.$$

<sup>10</sup>Recall that  $m_t = 2^{\log m - t}$  and  $S_\alpha = \{1, \dots, \Delta(\lambda)\} \cup \{\Delta(\lambda) + \alpha\} \subseteq [m']$ .

Note that the RAM proof  $\pi_{j,k}^{(t+1)}$  certifies the correctness of the  $k$ 'th element of  $c_j^{(t+1)}$  w.r.t.  $(\mathbf{h}, \mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_t, \mathbf{pk})$ .

7. For  $k \in [m']$ , let  $c_k \triangleq c_k^{(\log m)}$  and  $\pi_k \triangleq \pi_k^{(\log m)}$ .
8. Note that

$$(c_k)_{k \in [m']} = \sum_{j=1}^m (\mathbf{ct}_j)_{\mathbf{c}_1} = \text{LinEval}(\mathbf{pk}, (\mathbf{ct}_1)_{\mathbf{c}_1}, \dots, (\mathbf{ct}_m)_{\mathbf{c}_1}) = \mathbf{ct}_{\mathbf{c}_1}$$

where addition is coordinate-wise over  $\mathbb{F}_q$ , and

$$\mathbf{h}_{\log m} = \text{SEH.Hash}(\mathbf{hk}_{\log m}, (c_k, \pi_k)_{k \in [m']}).$$

9. Compute

$$\text{seBARG}_0.\pi = \text{seBARG}_0.\mathcal{P} \left( \begin{array}{l} \text{instances} = (\mathbf{pk}, \mathbf{v}_k, \mathbf{hk}, \mathbf{h}, \{\mathbf{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m}, k)_{k \in [m']}, \\ \text{witnesses} = (c_k, \pi_k, \rho_k)_{k \in [m']} \end{array} \right)$$

where  $\text{seBARG}_0$  is defined w.r.t. the NP language  $\mathcal{L}_0$  that has instances of length

$$n' = \left| (\mathbf{pk}, \mathbf{v}_k, \mathbf{hk}, \mathbf{h}, \{\mathbf{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m}, k) \right|,$$

and a valid witness  $(c_k, \pi_k, \rho_k)$  satisfies the following conditions:

- (a)  $\rho_k$  is a valid opening of  $(c_k, \pi_k)$  w.r.t.  $(\mathbf{hk}_{\log m}, \mathbf{h}_{\log m})$ . Namely,

$$\text{SEH.Verify}(\mathbf{hk}_{\log m}, \mathbf{h}_{\log m}, J_k, (c_k, \pi_k), \rho_k) = 1,$$

where  $J_k$  are the coordinates corresponding to  $(c_k, \pi_k)$  in  $(c_k, \pi_k)_{k \in [m']}$ .

- (b)  $\pi_k$  is a valid proof. Namely,

$$\text{RAM}_1.\mathcal{V}(\text{RAM}_1.\text{crs}, \mathbf{hk}_{\log m}, \mathbf{h}_{\log m}, (\mathbf{hk}, \mathbf{h}, \{\mathbf{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m-1}, \mathbf{pk}, 1, k, c_k), \pi_k) = 1.$$

- (c)  $c_k$  is consistent with  $\mathbf{v}_k$ . Namely, if  $k \in [m' - m]$ , then

$$\mathbf{v}_k = c_k$$

and if  $k > m' - m$ , then

$$\mathbf{v}_k = \text{Compress}_2(c_k).$$

10. Set  $\pi = (\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{\log m}, \text{seBARG}_0.\pi)$ .
11. Output  $(\mathbf{v}, \text{rt} = (\mathbf{h}, \pi))$ .

$\text{Validate}(\text{crs}_s, \mathbf{v}, \text{rt}) \rightarrow 0/1$ . This poly-time algorithm does the following:

1. Parse

$$\text{crs}_s = \left( \text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs} \right),$$

parse  $\text{rt} = (\text{h}, \pi)$  where

$$\pi = (\text{h}_0, \text{h}_1, \dots, \text{h}_{\log m}, \text{seBARG}_0.\pi),$$

and parse

$$\mathbf{v} = (\mathbf{v}_1, \dots, \mathbf{v}_{m'})$$

where recall that  $\mathbf{v}_j \in \mathbb{F}_q$  for every  $j \in [m' - m]$ , and  $\mathbf{v}_j \in \{0, 1\}$  for every  $j \in [m' - m + 1, m']$ .

2. Output 1 if and only if

$$\text{seBARG}_0.\mathcal{V} \left( \text{seBARG}_0.\text{crs}, \left( \text{pk}, \mathbf{v}_k, \text{hk}, \text{h}, \{\text{hk}_t, \text{h}_t\}_{t=0}^{\log m}, k \right)_{k \in [m']}, \text{seBARG}_0.\pi \right) = 1.$$

$\text{Open}(\text{crs}_\ell, \text{crs}_s, \mathbf{x}, i) \rightarrow (x_i, \rho)$ . This poly-time algorithm does the following:

1. Parse  $\text{crs}_\ell = \text{hk}' = (\text{pk}, \text{hk}'_1, \dots, \text{hk}'_m)$  and

$$\text{crs} = \left( \text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs} \right).$$

2. Compute

$$(\text{ct}_1, \dots, \text{ct}_m) = \text{SEH.Hash}(\text{hk}', \mathbf{x}).$$

3. Compute

$$(b, \rho'_1, \dots, \rho'_m) = \text{SEH.Open}(\text{hk}', \mathbf{x}, i).$$

Note that  $(\rho'_1, \dots, \rho'_m)$  is an opening of  $x_i$  w.r.t. the hash value  $(\text{ct}_1, \dots, \text{ct}_m)$ . Importantly, for every  $j \in [m]$ , one can verify  $\rho'_j$  given only  $(\text{pk}, \text{hk}'_j, \text{ct}_j)$ . Namely,

$$\text{SEH.Verify}((\text{pk}, \text{hk}'_j), \text{ct}_j, i, b, \rho'_j) = 1.$$

4. For every  $j \in [m]$  let

$$o_j = \text{SEH.Open}(\text{hk}, (\text{ct}_1, \dots, \text{ct}_m), [(j-1) \cdot n + 1, j \cdot n])$$

and let

$$o'_j = \text{SEH.Open}(\text{hk}, (\text{hk}'_1, \dots, \text{hk}'_m), [(j-1) \cdot n + 1, j \cdot n])$$

where  $n = |\text{ct}_j|$ .

Note that  $o_j$  is an opening of  $\text{ct}_j$  w.r.t. the hash value  $\text{h} = \text{SEH.Hash}(\text{hk}, (\text{ct}_1, \dots, \text{ct}_m))$ , and  $o'_j$  is an opening of  $\text{hk}'_j$  w.r.t. the hash value  $\text{h}' = \text{SEH.Hash}(\text{hk}, (\text{hk}'_1, \dots, \text{hk}'_m))$ .

5. Compute

$$\text{seBARG}_1.\pi = \text{seBARG}_1.\mathcal{P} \left( \text{seBARG}_1.\text{crs}, \begin{array}{l} \text{instances} = (\text{hk}, \text{pk}, \text{h}', \text{h}, i, b, j)_{j \in [m]}, \\ \text{witnesses} = (\text{ct}_j, \text{hk}'_j, \rho'_j, o_j, o'_j)_{j \in [m]} \end{array} \right)$$

where  $\text{seBARG}_1$  is defined w.r.t. the NP language  $\mathcal{L}_1$  that has instances of length

$$n'' = |(\text{hk}, \text{pk}, \text{h}', \text{h}, i, b, j)|$$

and a valid witness  $(\text{ct}_j, \text{hk}'_j, \rho'_j, o_j, o'_j)$  satisfies the following conditions:

(a)  $\rho'_j$  is a valid opening of  $b$  w.r.t.  $(\text{pk}, \text{hk}'_j, \text{ct}_j)$ . Namely,

$$\text{SEH.Verify}((\text{pk}, \text{hk}'_j), \text{ct}_j, i, b, \rho'_j) = 1.$$

(b)  $o_j$  is a valid opening of  $\text{ct}_j$  w.r.t.  $(\text{hk}, \text{h})$ . Namely,

$$\text{SEH.Verify}(\text{hk}, \text{h}, [(j-1) \cdot n + 1, j \cdot n], \text{ct}_j, o_j) = 1.$$

(c)  $o'_j$  is a valid opening of  $\text{hk}'_j$  w.r.t.  $(\text{hk}, \text{h}')$ . Namely,

$$\text{SEH.Verify}(\text{hk}, \text{h}', [(j-1) \cdot n + 1, j \cdot n], \text{hk}'_j, o'_j) = 1.$$

6. Output  $(b, \rho = \text{seBARG}_1.\pi)$ .

$\text{Verify}(\text{crs}_s, \text{rt}, i, b, \rho) \rightarrow 0/1$ . This poly-time algorithm does the following:

1. Parse

$$\text{crs}_s = \left( \text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs} \right),$$

and parse  $\text{rt} = (\text{h}, \pi)$ ,  $\rho = \text{seBARG}_1.\pi$ .

2. Output

$$\text{seBARG}_1.\mathcal{V}(\text{seBARG}_1.\text{crs}, (\text{hk}, \text{pk}, \text{h}', \text{h}, i, b, j)_{j \in [m]}, \text{seBARG}_1.\pi).$$

$\text{Extract}(\text{td}, \mathbf{v}) \rightarrow \{0, 1\}^m$ . This poly-time algorithm does the following:

1. Output  $\text{Dec}(\text{td}, \mathbf{v})$ .<sup>11</sup>

### 4.3 Analysis

**Theorem 4.2.** *Assuming hardness of the LWE problem, the scheme defined in Section 4.2 is a rate-1 fully-local somewhere extractable hash family as in Definition 4.1.*

*Proof.* In what follows we show that our construction satisfies all the desired properties.

**Efficiency.** Follows immediately from the efficiency of the underlying primitives, namely, the efficiency of the underlying rate-1 linearly homomorphic encryption scheme, the SEH scheme, the RAM SNARGs, and the seBARG schemes. If  $I = \{i^* + 1, \dots, i^* + m\}$  for some  $i^* \in [N]$ , then  $\text{Gen}$  runs in time  $\text{poly}(\lambda, \log N)$  (see Remark 4.3).

**Index hiding.** Follows immediately from the index hiding property of the underlying SEH hash family (see Definition 3.1).

**Completeness.** Follows immediately from the completeness property of the underlying  $\text{RAM}_0, \text{RAM}_1$  schemes (Definition 3.7), the completeness of the underlying  $\text{seBARG}_0, \text{seBARG}_1$  schemes (Definition 3.3), and the opening completeness of the underlying SEH family (Definition 3.1).

<sup>11</sup>Recall that the SEH hash family we used has the property that the trapdoor key  $\text{td}$  is the secret key corresponding to the underlying encryption scheme (see Lemma 3.2).

**Somewhere statistically binding w.r.t. opening.** Fix any (possibly all-powerful) adversary  $\mathcal{A}$ , any polynomial  $N = N(\lambda)$ , subset  $I \subseteq [N]$ , and index  $i^* \in I$ . We need to prove that there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{Verify}(\text{crs}_s, \text{rt}, i^*, b, \rho) = 1 \\ \wedge \text{Validate}(\text{crs}_s, \text{v}, \text{rt}) = 1 \\ \wedge b \neq b_{i^*} \end{array} : \begin{array}{l} (\text{crs}_\ell, \text{crs}_s, \text{td}') \leftarrow \text{Gen}(1^\lambda, N, I), \\ (\text{v}, \text{rt}, b, \rho) \leftarrow \mathcal{A}(\text{crs}_\ell, \text{crs}_s), \\ (b_i)_{i \in I} = \text{Extract}(\text{td}, \text{v}) \end{array} \right] \leq \mu(\lambda). \quad (3)$$

Parse  $I = \{i_1, \dots, i_m\}$  and suppose that  $i^* = i_\alpha$ . Let  $\text{Gen}_\alpha$  be identical to  $\text{Gen}$  except that it sets  $\alpha$  as indicated, instead of arbitrarily, and sets  $\beta = \alpha$ . In addition,  $\text{Gen}_\alpha$  does not output only  $\text{td}'$  as the trapdoor, but also outputs other trapdoors generated during the  $\text{Gen}$  algorithm. Namely, it outputs

$$\text{td}^* = (\text{td}', \text{td}, \text{seBARG}_0.\text{td}, \text{seBARG}_1.\text{td}).$$

Below we define an extraction algorithm  $\mathcal{E}$ , which calls as subroutines the polynomial time extractors  $\text{SEHash.Extract}$ ,  $\text{seBARG}_0.\mathcal{E}$ , and  $\text{seBARG}_1.\mathcal{E}$ .  $\mathcal{E}$  takes as input the trapdoors  $\text{td}^*$ , short crs  $\text{crs}_s$ , hash value  $(\text{v}, \text{rt})$ , bit  $b$ , and opening  $\rho$  and does the following:

1. Parse  $\text{td}^* = (\text{td}', \text{td}, \{\text{td}_t\}_{t=0}^{\log m}, \text{seBARG}_0.\text{td}, \text{seBARG}_1.\text{td})$ .
2. Parse  $\text{crs}_s = (\text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs})$ .
3. Parse  $\text{rt} = (\text{h}, \pi)$ ,  $\pi = (\text{h}_0, \dots, \text{h}_{\log m}, \text{seBARG}_0.\pi)$ ,  $\rho = \text{seBARG}_1.\pi$ .
4. Compute  $\text{hk}'_\alpha = \text{SEH.Extract}(\text{td}, \text{h}')$  and  $\text{ct}_\alpha = \text{SEH.Extract}(\text{td}, \text{h})$ .
5. Compute  $(c_k, \pi_k)_{k \in S_\alpha} = \text{SEH.Extract}(\text{td}_{\log m}, \text{h}_{\log m})$ .
6. Compute  $\omega_0 = \text{seBARG}_0.\mathcal{E}(\text{seBARG}_0.\text{td}, (\text{pk}, \text{v}_k, \text{hk}, \text{h}, \{\text{hk}_t, \text{h}_t\}_{t=0}^{\log m}, k)_{k \in [m']}, \text{seBARG}_0.\pi)$ .
7. Compute  $\omega_1 = \text{seBARG}_1.\mathcal{E}(\text{seBARG}_1.\text{td}, (\text{hk}, \text{pk}, \text{h}', \text{h}, i^*, b, j)_{j \in [m]}, \text{seBARG}_1.\pi)$ .
8. Output  $(\text{hk}'_\alpha, \text{ct}_\alpha, (c_k, \pi_k)_{k \in S_\alpha}, \omega_0, \omega_1)$ .

We define the following experiment:

EXPT $_\alpha$
$(\text{crs}_\ell, \text{crs}_s, \text{td}^*) \leftarrow \text{Gen}_\alpha(1^\lambda, N, I)$
$(\text{v}, \text{rt}, b, \rho) \leftarrow \mathcal{A}(\text{crs}_\ell, \text{crs}_s)$
$(b_i)_{i \in I} = \text{Extract}(\text{td}, \text{v})$
$(\text{hk}'_\alpha, \text{ct}_\alpha, (c_k, \pi_k)_{k \in S_\alpha}, \omega_0, \omega_1) = \mathcal{E}(\text{td}^*, \text{crs}_s, \text{v}, \text{rt}, b, \rho)$

By the index hiding property of the underlying SEH family and the underlying seBARG schemes, to prove Equation (3) it suffices to prove that there exists a negligible function  $\mu_0$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_\alpha} \left[ b \neq b_{i_\alpha} \wedge \text{Verify}(\text{crs}_s, \text{rt}, i_\alpha, b, \rho) = 1 \wedge \text{Validate}(\text{crs}_s, \text{v}, \text{rt}) = 1 \right] \leq \mu_0(\lambda). \quad (4)$$



Recall that  $\text{Verify}(\text{crs}_s, \text{rt}, i_\alpha, b, \rho) = 1 \wedge \text{Validate}(\text{crs}_s, \mathbf{v}, \text{rt}) = 1$  if and only if

$$\text{seBARG}_0.\mathcal{V} \left( \text{seBARG}_0.\text{crs}, \left( \text{pk}, \mathbf{v}_k, \text{hk}, \mathbf{h}, \{\text{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m}, k \right)_{k \in [m]}, \text{seBARG}_0.\pi \right) = 1$$

and

$$\text{seBARG}_1.\mathcal{V} \left( \text{seBARG}_1.\text{crs}, (\text{hk}, \text{pk}, \mathbf{h}', \mathbf{h}, i_\alpha, b, j)_{j \in [m]}, \text{seBARG}_1.\pi \right) = 1.$$

In what follows, we parse

$$\omega_0 = (c'_k, \pi'_k, \rho_k)_{k \in S_\alpha} \text{ and } \omega_1 = (\text{ct}'_\alpha, \text{hk}''_\alpha, \rho'_\alpha, o_\alpha, o'_\alpha).$$

The somewhere argument of knowledge property of the underlying  $\text{seBARG}_0, \text{seBARG}_1$  schemes (Definition 3.3) implies that to prove Equation (4) it suffices to prove that there exists a negligible function  $\mu_1$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_\alpha} \left[ \begin{array}{l} b \neq b_{i_\alpha} \\ \wedge \forall k \in S_\alpha : (c'_k, \pi'_k, \rho_k) \text{ is a valid witness for} \\ \quad \left( \text{pk}, \mathbf{v}_k, \text{hk}, \mathbf{h}, \{\text{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m}, k \right) \in \mathcal{L}_0 \\ \wedge (\text{ct}'_\alpha, \text{hk}''_\alpha, \rho'_\alpha, o_\alpha, o'_\alpha) \text{ is a valid witness for } (\text{hk}, \text{pk}, \mathbf{h}', \mathbf{h}, i_\alpha, b, \alpha) \in \mathcal{L}_1 \end{array} \right] \leq \mu_1(\lambda). \quad (5)$$

Recall that  $(\text{ct}'_\alpha, \text{hk}''_\alpha, \rho'_\alpha, o_\alpha, o'_\alpha)$  is a valid witness for  $(\text{hk}, \text{pk}, \mathbf{h}', \mathbf{h}, i_\alpha, b, \alpha) \in \mathcal{L}_1$  if and only if  $\rho'_\alpha, o_\alpha, o'_\alpha$  are valid openings for  $b, \text{ct}'_\alpha, \text{hk}''_\alpha$  respectively (see Item 5). The somewhere statistically binding w.r.t opening property of the underlying SEH family (Definition 3.1) implies that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_\alpha} \left[ \begin{array}{l} (\text{ct}'_\alpha, \text{hk}''_\alpha, \rho'_\alpha, o_\alpha, o'_\alpha) \text{ is a valid witness for } (\text{hk}, \text{pk}, \mathbf{h}', \mathbf{h}, i_\alpha, b, \alpha) \in \mathcal{L}_1 \\ \wedge b \neq \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \end{array} \right] \leq \text{negl}(\lambda). \quad (6)$$

It remains to argue that there exists a negligible function  $\mu_2$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_\alpha} \left[ \begin{array}{l} \forall k \in S_\alpha : (c'_k, \pi'_k, \rho_k) \text{ is a valid witness for} \\ \quad \left( \text{pk}, \mathbf{v}_k, \text{hk}, \mathbf{h}, \{\text{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m}, k \right) \in \mathcal{L}_0 \\ \wedge \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \neq b_{i_\alpha} \end{array} \right] \leq \mu_2(\lambda).$$

Recall that  $(c'_k, \pi'_k, \rho_k)$  is a valid witness for  $(\text{pk}, \mathbf{v}_k, \text{hk}, \mathbf{h}, \{\text{hk}_t, \mathbf{h}_t\}_{t=0}^{\log m}, k) \in \mathcal{L}_0$  if and only if (1)  $\rho_k$  is a valid opening of  $(c'_k, \pi'_k)$  w.r.t.  $(\text{hk}_{\log m}, \mathbf{h}_{\log m})$ , (2)  $\pi'_k$  is a valid RAM proof, and (3)  $c'_k$  is consistent with  $\mathbf{v}_k$  (see Item 9). Thus by the somewhere statistically binding w.r.t. opening property of the underlying SEH family (Definition 3.1) to prove Equation (5) it suffices to argue that there exists a negligible function  $\mu_3$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_\alpha} \left[ \begin{array}{l} \forall k \in S_\alpha : \pi_k \text{ is a valid proof} \\ \wedge \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \neq \text{CompDec}(\text{sk}_c, (\text{Compress}_2(c_k))_{k \in S_\alpha}) \end{array} \right] \leq \mu_3(\lambda). \quad (7)$$

since if  $c_k$  is consistent with  $\mathbf{v}_k$  for all  $k \in S_\alpha$ , then

$$\text{CompDec}(\text{sk}_c, (\text{Compress}_2(c_k))_{k \in S_\alpha}) = \text{CompDec}(\text{sk}_c, (\mathbf{v}_k)_{k \in S_\alpha}) = b_{i_\alpha}.$$

For any  $\beta \in [m]$ , let  $\text{Gen}_{\alpha,\beta}$  be identical to  $\text{Gen}_\alpha$  except that it sets  $\beta$  as indicated, instead of setting  $\beta = \alpha$ . In addition,  $\text{Gen}_{\alpha,\beta}$  outputs

$$\text{td}^* = \left( \text{td}', \text{td}, \{\text{td}_t\}_{t=0}^{\log m} \right)$$

as the trapdoor. Below we define another extraction algorithm  $\mathcal{E}'$ , which calls as subroutines the polynomial time extractor  $\text{SEHash.Extract}$ .  $\mathcal{E}'$  takes as input the trapdoors  $\text{td}^*$ , short crs  $\text{crs}_s$ , and short hash value  $\text{rt}$  and does the following:

1. Parse  $\text{td}^* = \left( \text{td}', \text{td}, \{\text{td}_t\}_{t=0}^{\log m} \right)$ .
2. Parse  $\text{crs}_s = \left( \text{pk}, \text{pk}_c, \text{hk}, \text{h}', \{\text{hk}_t\}_{t=0}^{\log m}, \text{RAM}_0.\text{crs}, \text{RAM}_1.\text{crs}, \text{seBARG}_0.\text{crs}, \text{seBARG}_1.\text{crs} \right)$ .
3. Parse  $\text{rt} = (\text{h}, \pi)$ ,  $\pi = (\text{h}_0, \dots, \text{h}_{\log m}, \text{seBARG}_0.\pi)$ .
4. Compute  $\text{ct}_\beta = \text{SEH.Extract}(\text{td}, \text{h})$ .
5. For  $t \in [0, \log m]$ : Compute  $\left( c_{j,k}^{(t)}, \pi_{j,k}^{(t)} \right)_{j \in J'_{t,\beta}, k \in S_\alpha} = \text{SEH.Extract}(\text{td}_t, \text{h}_t)$ .
6. Output  $\left( \text{ct}_\beta, \left\{ \left( c_{j,k}^{(t)}, \pi_{j,k}^{(t)} \right)_{j \in J'_{t,\beta}, k \in S_\alpha} \right\}_{t=0}^{\log m} \right)$ .

We define the experiment  $\text{EXPT}_{\alpha,\beta}$  identically to  $\text{EXPT}_\alpha$  except it calls  $\text{Gen}_{\alpha,\beta}$  and  $\mathcal{E}'$  instead of  $\text{Gen}_\alpha$  and  $\mathcal{E}$ . We note that the partial-input soundness property of the underlying  $\text{RAM}_1$  scheme (applied  $\log m - t$  times) implies that there exists a negligible function  $\nu_1$  such that for every  $\lambda \in \mathbb{N}, t \in [0, \log m - 1], k \in S_\alpha$ , and  $j \in J'_{t,\beta}$ ,

$$\Pr_{\text{EXPT}_{\alpha,\beta}} \left[ \begin{array}{l} \pi_k \text{ is a valid RAM proof} \\ \wedge \pi_{j,k}^{(t)} \text{ is not a valid RAM proof} \end{array} \right] \leq (\log m - t) \cdot \nu_1(\lambda) \leq \text{negl}(\lambda).$$

Recall that for  $k \in S_\alpha$ ,  $c_{1,k}^{(\log m)}$  is computed by summing  $c_{1,k}^{(0)}, \dots, c_{m,k}^{(0)}$  in a tree-based fashion, and  $\mathcal{E}'$  extracts the one path in the tree which the hash keys  $\text{hk}_0, \dots, \text{hk}_{\log m}$  are somewhere extractable on. For  $t \in [0, \log m]$ , let  $\alpha_t = \lceil \frac{\alpha}{2^t} \rceil$  (note that  $\alpha_0 = \alpha$  and  $\alpha_{\log m} = 1$ ); then in a correctly computed tree we have that

$$\text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{\alpha_t,k}^{(t)} \right) \right)_{k \in S_\alpha} \right) = \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \quad (8)$$

and

$$\forall j \in [m_t] \text{ with } j \neq \alpha_t : \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{j,k}^{(t)} \right) \right)_{k \in S_\alpha} \right) = 0. \quad (9)$$

In what follows, we prove the correctness of  $c_{1,k}^{(\log m)}$  (assuming the validity of  $\pi_{1,k}^{(\log m)}$ ) by inductively proving eq. (9) in claim 4.2.1 and eq. (8) in claim 4.2.2.

**Claim 4.2.1.** *There exists a negligible function  $\nu_2$  such that for every  $\lambda \in \mathbb{N}, t \in [0, \log m - 1]$ , and  $j \in J'_{t,\beta}$  with  $j \neq \alpha_t$ ,*

$$\Pr_{\text{EXPT}_{\alpha,\beta}} \left[ \begin{array}{l} \left( \pi_{j,k}^{(t)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \wedge \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{j,k}^{(t)} \right) \right)_{k \in S_\alpha} \right) \neq 0 \end{array} \right] \leq \nu_2(\lambda).$$

**Proof of Claim 4.2.1.** We proceed by induction on  $t$ .

- **Base case** ( $t = 0$ ). By the index hiding property of the underlying SEH family, it suffices to bound the desired probability over  $\text{EXPT}_{\alpha,j}$ . Recall that for  $k \in S_\alpha$ ,  $\pi_{j,k}^{(0)}$  certifies that  $c_{j,k}^{(0)}$  is the  $k$ -th element of  $\text{Compress}_1(\text{pk}_c, \text{ct}_j, m, j)$ , and that  $\text{Compress}_1(\text{pk}_c, \text{ct}_j, m, j)$  is a ciphertext encrypting the vector  $\text{SEH.Extract}(\text{td}', \text{ct}_j) \cdot e_j$ , which is 0 everywhere except at coordinate  $j$ . The correctness of the encryption scheme implies that for all  $j \neq \alpha$ ,

$$\text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( \left( \text{Compress}_1(\text{pk}_c, \text{ct}_j, m, j) \right)_k \right)_{k \in S_\alpha} \right) \right) = 0,$$

except with negligible probability. Thus the partial-input soundness property of the underlying  $\text{RAM}_0$  scheme implies that for all  $\lambda \in \mathbb{N}$  and  $j \neq \alpha$ ,

$$\Pr_{\text{EXPT}_{\alpha,j}} \left[ \begin{array}{l} \pi_{j,k}^{(0)} \text{ is valid} \\ \wedge \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{j,k}^{(0)} \right) \right)_{k \in S_\alpha} \right) \neq 0 \end{array} \right] \leq \text{negl}(\lambda).$$

- **Inductive step.** Set  $\beta' \in [m]$  so  $2j - 1, 2j \in J'_{t-1, \beta'}$  and  $j \in J'_{t, \beta'}$ . By the index hiding property of the underlying SEH family, it suffices to bound the desired probability over  $\text{EXPT}_{\alpha, \beta'}$ . The inductive hypothesis implies that for every  $\lambda \in \mathbb{N}$  and  $j \neq \alpha_t$ ,

$$\Pr_{\text{EXPT}_{\alpha, \beta'}} \left[ \begin{array}{l} \left( \pi_{2j-1, k}^{(t-1)}, \pi_{2j, k}^{(t-1)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \wedge \left( \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{2j-1, k}^{(t-1)} \right) \right)_{k \in S_\alpha} \right) \neq 0 \right. \\ \left. \vee \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{2j, k}^{(t-1)} \right) \right)_{k \in S_\alpha} \right) \neq 0 \right) \end{array} \right] \leq \text{negl}(\lambda).$$

since  $j \neq \lceil \frac{\alpha}{2^t} \rceil \implies 2j - 1 \neq \lceil \frac{\alpha}{2^{t-1}} \rceil = \alpha_{t-1}$  and  $2j \neq \lceil \frac{\alpha}{2^{t-1}} \rceil = \alpha_{t-1}$ . Recall that  $\pi_{j,k}^{(t)}$  certifies that  $c_{j,k}^{(t)} = c_{2j-1, k}^{(t-1)} + c_{2j, k}^{(t-1)}$ , where addition is over the finite field  $\mathbb{F}_q$ , and that  $\pi_{2j-1, k}^{(t-1)}, \pi_{2j, k}^{(t-1)}$  are valid RAM proofs. Thus the partial-input soundness property of the underlying  $\text{RAM}_1$  scheme and the correctness of the encryption scheme implies that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_{\alpha, \beta'}} \left[ \begin{array}{l} \left( \pi_{j,k}^{(t)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{j,k}^{(t)} \right) \right)_{k \in S_\alpha} \right) \neq 0 \end{array} \right] \leq \text{negl}(\lambda).$$

**Claim 4.2.2.** *There exists a negligible function  $\nu_3$  such that for every  $\lambda \in \mathbb{N}$ ,  $t \in [0, \log m]$ ,*

$$\Pr_{\text{EXPT}_{\alpha, \alpha}} \left[ \begin{array}{l} \left( \pi_{\alpha_t, k}^{(t)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \wedge \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{\lceil \frac{\alpha}{2^t} \rceil, k}^{(t)} \right) \right)_{k \in S_\alpha} \right) \neq \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \end{array} \right] \leq \nu_3(\lambda).$$

**Proof of Claim 4.2.2.** We proceed by induction on  $t$ .

- **Base case** ( $t = 0$ ). Recall that for  $k \in S_\alpha$ ,  $\pi_{\alpha,k}^{(0)}$  certifies that  $c_{\alpha,k}^{(0)}$  is the  $k$ -th element of  $\text{Compress}_1(\text{pk}_c, \text{ct}_\alpha, m, \alpha)$ . Thus the correctness of the encryption scheme and the partial-input soundness property of the underlying  $\text{RAM}_0$  scheme implies that for all  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_{\alpha,\alpha}} \left[ \begin{array}{l} \pi_{\alpha,k}^{(0)} \text{ is valid} \\ \wedge \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{\alpha,k}^{(0)} \right) \right)_{k \in S_\alpha} \right) \neq \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \end{array} \right] \leq \text{negl}(\lambda).$$

- **Inductive step.** The inductive hypothesis implies that

$$\Pr_{\text{EXPT}_{\alpha,\alpha}} \left[ \begin{array}{l} \left( \pi_{\alpha_{t-1},k}^{(t-1)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \wedge \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{\alpha_{t-1},k}^{(t-1)} \right) \right)_{k \in S_\alpha} \right) \neq \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \end{array} \right] \leq \text{negl}(\lambda).$$

Let  $\alpha'_{t-1} = \alpha_{t-1} \pm 1$  be the index of the other child of  $(c_{\alpha_t,k}^{(t)}, \pi_{\alpha_t,k}^{(t)})$ . Claim 4.2.1 implies that

$$\Pr_{\text{EXPT}_{\alpha,\alpha}} \left[ \begin{array}{l} \left( \pi_{\alpha'_{t-1},k}^{(t-1)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \wedge \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{\alpha'_{t-1},k}^{(t-1)} \right) \right)_{k \in S_\alpha} \right) \neq 0 \end{array} \right] \leq \text{negl}(\lambda).$$

Recall that  $\pi_{\alpha_t,k}^{(t)}$  certifies that  $c_{\alpha_t,k}^{(t)} = c_{\alpha_{t-1},k}^{(t-1)} + c_{\alpha'_{t-1},k}^{(t-1)}$ , where addition is over the finite field  $\mathbb{F}_q$ , and that  $\pi_{\alpha_{t-1},k}^{(t-1)}, \pi_{\alpha'_{t-1},k}^{(t-1)}$  are valid RAM proofs. Thus the partial-input soundness property of the underlying  $\text{RAM}_1$  scheme and the correctness of the encryption scheme implies that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr_{\text{EXPT}_{\alpha,\alpha}} \left[ \begin{array}{l} \left( \pi_{\alpha_t,k}^{(t)} \right)_{k \in S_\alpha} \text{ are valid RAM proofs} \\ \text{CompDec} \left( \text{sk}_c, \left( \text{Compress}_2 \left( c_{\alpha_t,k}^{(t)} \right) \right)_{k \in S_\alpha} \right) \neq \text{SEH.Extract}(\text{td}', \text{ct}_\alpha) \end{array} \right] \leq \text{negl}(\lambda).$$

Claim 4.2.2 for  $t = \log m$  implies eq. (7), so we are done.  $\square$

**Extraction correctness.** Follows immediately from the correctness of the encryption scheme (Definition 3.10).  $\square$

## 5 Rate-1 seBARGs

In this section we construct a rate-1 seBARG (Definitions 3.3 and 3.6) for any NP language  $\mathcal{L} = \{\mathcal{L}_n\}_{n \in \mathbb{N}}$  with witnesses of length  $m = m(n)$ . Our construction of a rate-1 seBARG uses the following ingredients:

1. A rate-1 fSEH family (see Section 4)

(H.Gen, H.Hash, H.Extract, H.Digest, H.Open, H.Verify).

2. A flexible RAM SNARG scheme with partial input soundness (see Section 3.3)

$$(\text{RAM.Gen}, \text{RAM.Digest}, \text{RAM.P}, \text{RAM.V})$$

as in Theorem 3.8 w.r.t. the RAM machine  $\mathcal{R}$  which takes an implicit input  $(\omega_1, \dots, \omega_k)$  and an explicit input  $(x_i, i)$  and outputs 1 if and only if  $\omega_i$  is a valid witness for  $x_i \in \mathcal{L}$ . Let  $T$  denote the run-time of  $\mathcal{R}$ .

3. An index seBARG scheme with proof length  $|\text{seB.}\pi| = m \cdot \text{poly}(\lambda, \log k)$  as in Theorem 3.5 (see Section 3.2)

$$(\text{seB.Gen}, \text{seB.P}, \text{seB.V})$$

for the NP language

$$\mathcal{L}' = \left\{ (\text{RAM.crs}, \text{H.crs}, \text{rt}, x_i, i) \mid \begin{array}{l} \exists \text{RAM.}\pi \in \{0, 1\}^{\text{poly}(\lambda)} \text{ s.t.} \\ \text{RAM.V}(\text{RAM.crs}, \text{H.crs}, \text{rt}, (x_i, i), \text{RAM.}\pi) = 1 \end{array} \right\}.$$

We are now ready to construct our rate-1 seBARG for  $\mathcal{L}$ .

$\text{Gen}(1^\lambda, k, n, i^*) \rightarrow (\text{crs}, \text{td})$ . This poly-time algorithm does the following:

1. Generate  $(\text{H.crs}, \text{H.td}) \leftarrow \text{H.Gen}(1^\lambda, k \cdot m, I)$  where  $I = \{(i^* - 1) \cdot m + 1, \dots, i^* \cdot m\}$ .<sup>12</sup>
2. Generate  $\text{RAM.crs} \leftarrow \text{RAM.Gen}(1^\lambda, T)$ .
3. Generate  $(\text{seB.crs}, \text{seB.td}) \leftarrow \text{seB.Gen}(1^\lambda, k, n', i^*)$   
where  $n' = |(\text{RAM.crs}, \text{H.crs}, \text{rt}, i, x_i)|$ .
4. Output  $\text{crs} = (\text{H.crs}, \text{RAM.crs}, \text{seB.crs})$  and  $\text{td} = (\text{H.td}, \text{seB.td})$ .

$\mathcal{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \rightarrow \Pi$ . This poly-time algorithm does the following:

1. Parse  $\text{crs} = (\text{H.crs}, \text{RAM.crs}, \text{seB.crs})$ .
2. Compute  $(v, \text{rt}) = \text{H.Hash}(\text{H.crs}, (\omega_1, \dots, \omega_k))$ .
3. For every  $i \in [k]$  let  $\text{RAM.}\pi_i = \text{RAM.P}(\text{RAM.crs}, \text{H.crs}, (\omega_1, \dots, \omega_k), (x_i, i))$ .
4. Compute  $\text{seB.}\pi \leftarrow \text{seB.P} \left( \text{seB.crs}, \begin{array}{l} \text{instances} = (\text{RAM.crs}, \text{H.crs}, \text{rt}, x_i, i)_{i \in [k]}, \\ \text{witnesses} = (\text{RAM.}\pi_i)_{i \in [k]} \end{array} \right)$ .
5. Output  $\Pi = (v, \text{rt}, \text{seB.}\pi)$ .

$\mathcal{V}(\text{crs}, x_1, \dots, x_k, \Pi) \rightarrow 0/1$ . This poly-time algorithm does the following:

1. Parse  $\text{crs} = (\text{H.crs}, \text{RAM.crs}, \text{seB.crs})$ .
2. Parse  $\Pi = (v, \text{rt}, \text{seB.}\pi)$ .
3. Output 1 if and only if

$$\text{H.Validate}(\text{H.crs}, v, \text{rt}) = 1$$

and

$$\text{seB.V}(\text{seB.crs}, (\text{RAM.crs}, \text{H.crs}, \text{rt}, x_i, i)_{i \in [k]}, \text{seB.}\pi) = 1.$$

**Theorem 5.1.** *Assuming hardness of the LWE problem,  $(\text{Gen}, \mathcal{P}, \mathcal{V})$  is a rate-1 seBARG as in Definitions 3.3 and 3.6.*

*Proof.* In what follows we prove that our construction satisfies all the desired properties.

<sup>12</sup>Recall that a fSEH hash family has the property that if  $I$  consists of consecutive indices (which is the case here) then  $\text{Gen}$  runs in time  $\text{poly}(\lambda, \log k, \log m)$ , and outputs a single (shrot) crs, denoted above by  $\text{H.crs}$ .

**Efficiency.** We first note that our construction satisfies the efficiency requirement.

1. The fact that  $\text{Gen}$  is a poly-time algorithm follows from the efficiency guarantee of the key generation algorithm of the underlying primitives: the fISEH family, the RAM delegation scheme (Definition 3.7), and the seB scheme (Definition 3.3).
2. The fact that the proof  $\Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)$  is of length  $m + \text{poly}(\lambda)$  follows from the fact that the underlying fISEH family has rate-1, which implies that  $|\mathbf{v}| = m + \text{poly}(\lambda)$ ,  $|\text{rt}| \leq \text{poly}(\lambda)$ , and the time to verify an opening is  $\text{poly}(\lambda)$ , together with the efficiency guarantees of the underlying RAM and seB schemes, which ensure that  $|\text{RAM}.\pi_i| \leq \text{poly}(\lambda)$  and thus  $|\text{seB}.\pi| \leq \text{poly}(\lambda)$ .

Moreover,  $\Pi$  can be generated in polynomial time, which follows from the polynomial running time of the underlying algorithms.

**Index hiding.** The index hiding condition follows directly from the index hiding property of the underlying fISEH family (Definition 4.1) and the seB scheme (Definition 3.3).

**Completeness.** The completeness condition follows immediately from the completeness condition of the underlying RAM delegation scheme (Definition 3.7), the completeness condition of the underlying seB scheme (Definition 3.3), and the validation/opening completeness condition of the underlying fISEH scheme (Definition 4.1).

**Somewhere argument of knowledge.** We define a PPT extractor  $\mathcal{E}$  that takes as input a tuple  $(\text{td}, (x_1, \dots, x_k), \Pi)$ , and does the following:

1. Parse  $\Pi = (\mathbf{v}, \text{rt}, \pi)$ , and  $\text{td} = (\text{H}.\text{td}, \text{seB}.\text{td})$ .
2. Output  $\omega = \text{H}.\text{Extract}(\text{H}.\text{td}, \mathbf{v})$ .

Suppose for the sake of contradiction that there exists a poly-size adversary  $\mathcal{A}$ , polynomials  $k = k(\lambda)$  and  $n = n(\lambda)$ , an index  $i^* \in [k]$ , and a non-negligible function  $\epsilon$ , such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, x_1, \dots, x_k, \Pi) = 1 \\ \wedge \omega_{i^*} \text{ is not a valid witness for } x_{i^*} \in \mathcal{L} \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*), \\ (x_1, \dots, x_k, \Pi) \leftarrow \mathcal{A}(\text{crs}), \\ \omega_{i^*} \leftarrow \mathcal{E}(\text{td}, x_1, \dots, x_k, \Pi) \end{array} \right] \geq \epsilon(\lambda). \quad (10)$$

We abbreviate  $\mathbf{x} = (x_1, \dots, x_k)$ . Recall that

$$\mathcal{V}(\text{crs} = (\text{H}.\text{crs}, \text{RAM}.\text{crs}, \text{seB}.\text{crs}), \mathbf{x}, \Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)) = 1$$

if and only if

$$\text{H}.\text{Validate}(\text{H}.\text{crs}, \mathbf{v}, \text{rt}) = 1$$

and

$$\text{seB}.\mathcal{V}(\text{seB}.\text{crs}, (\text{RAM}.\text{crs}, \text{H}.\text{crs}, \text{rt}, x_i, i)_{i \in [k]}, \text{seB}.\pi) = 1.$$

Thus, Equation (10) implies that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{H}.\text{Validate}(\text{H}.\text{crs}, \mathbf{v}, \text{rt}) = 1 \\ \wedge \text{seB}.\mathcal{V}(\text{seB}.\text{crs}, \\ \quad (\text{RAM}.\text{crs}, \text{H}.\text{crs}, \text{rt}, x_i, i)_{i \in [k]}, \text{seB}.\pi) = 1 \\ \wedge \omega_{i^*} \text{ is not a valid witness for } x_{i^*} \in \mathcal{L} \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*), \\ (\mathbf{x}, \Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)) = \mathcal{A}(\text{crs}) \end{array} \right] \geq \epsilon(\lambda). \quad (11)$$

Recall that  $\text{seB}.\pi$  certifies that for all  $i \in [k]$  there exists a short proof  $\text{RAM}.\pi_i$  such that  $\text{RAM}.\mathcal{V}(\text{RAM}.\text{crs}, \text{H}.\text{crs}_s, \text{rt}, (x_i, i), \text{RAM}.\pi_i) = 1$ . Let  $\text{seB}.\mathcal{E}$  be the PPT extractor associated with  $\text{seB}$ . Define a PPT extractor  $\mathcal{E}'$  which takes as input crses  $\text{crs} = (\text{H}.\text{crs}, \text{RAM}.\text{crs}, \text{seB}.\text{crs})$ , trapdoors  $\text{td} = (\text{H}.\text{td}, \text{seB}.\text{td})$ , instance  $\mathbf{x}$ , and proof  $\Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)$  and outputs

$$\text{RAM}.\pi^* = \text{seB}.\mathcal{E}(\text{seB}.\text{td}, (\text{RAM}.\text{crs}, \text{H}.\text{crs}, \text{rt}, x_i, i)_{i \in [k]}, \text{seB}.\pi).$$

By the somewhere argument of knowledge property of  $\text{seB}$  (Definition 3.3), there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{seB}.\mathcal{V}(\text{seB}.\text{crs}, \text{seB}.\mathbf{x}, \text{seB}.\pi) = 1 \\ \wedge \text{RAM}.\mathcal{V}(\text{RAM}.\text{crs}, \text{H}.\text{crs}, \\ \text{rt}, (x_{i^*}, i^*), \text{RAM}.\pi^*) \neq 1 \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*), \\ (\mathbf{x}, \Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)) = \mathcal{A}(\text{crs}) \\ \text{RAM}.\pi^* = \mathcal{E}'(\text{crs}, \text{td}, \mathbf{x}, \Pi) \end{array} \right] \leq \mu(\lambda) \quad (12)$$

Combining Equations (11) and (12) we conclude that there exists a non-negligible function  $\epsilon'(\lambda) = \epsilon(\lambda) - \mu(\lambda)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \omega_{i^*} \text{ is not a valid witness for } x_{i^*} \in \mathcal{L} \\ \wedge \text{H}.\text{Validate}(\text{H}.\text{crs}, \mathbf{v}, \text{rt}) = 1 \\ \wedge \text{RAM}.\mathcal{V}(\text{RAM}.\text{crs}, \text{H}.\text{crs}, \\ \text{rt}, (x_{i^*}, i^*), \text{RAM}.\pi^*) = 1 \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*), \\ (\mathbf{x}, \Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)) = \mathcal{A}(\text{crs}), \\ \text{RAM}.\pi^* = \mathcal{E}'(\text{crs}, \text{td}, \mathbf{x}, \Pi) \end{array} \right] \geq \epsilon'(\lambda). \quad (13)$$

Recall that the RAM machine  $\mathcal{R}$  associated with the RAM SNARG scheme takes as input  $(\omega_1, \dots, \omega_k), (x_i, i)$  and outputs 1 if and only if  $\omega_i$  is a valid witness for  $x_i \in \mathcal{L}$ . Equation (13) implies that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{R}((\omega_1, \dots, \omega_k), (x_{i^*}, i^*)) = 0 \\ \wedge \text{H}.\text{Validate}(\text{H}.\text{crs}, \mathbf{v}, \text{rt}) = 1 \\ \wedge \text{RAM}.\mathcal{V}(\text{RAM}.\text{crs}, \text{H}.\text{crs}, \\ \text{rt}, (x_{i^*}, i^*), \text{RAM}.\pi^*) = 1 \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, n, i^*), \\ (\mathbf{x}, \Pi = (\mathbf{v}, \text{rt}, \text{seB}.\pi)) = \mathcal{A}(\text{crs}), \\ \text{RAM}.\pi^* = \mathcal{E}'(\text{crs}, \text{td}, \mathbf{x}, \Pi), \\ \omega_{i^*} = \text{H}.\text{Extract}(\text{H}.\text{td}, \mathbf{v}), \\ \forall i \in [k] \setminus \{i^*\} : \omega_i = 0^m \end{array} \right] \geq \epsilon'(\lambda)$$

but this contradicts partial-input soundness of the RAM delegation scheme (Definition 3.7).  $\square$

## 6 Multi-Hop seBARGs

In this section, we extend the notion of a seBARG to the *multi-hop* setting, and describe a generic construction based on any rate-1 seBARG. In the multi-hop setting, seBARGs can be further batched with other seBARGs (or even a single instance-witness pair) succinctly. The number of *hops* (i.e., number of times seBARGs can be successively batched) can be any polynomial, and the batch size in each hop can be set arbitrarily. Each hop increases the proof size by an *additive*  $\text{poly}(\lambda)$  factor.

In Section 6.3 we define and construct a hashed version of a multi-hop seBARG where all the instances are hashed and are not given explicitly to the verifier. In Section 7 we show how to use a multi-hop seBARG to construct a multi-hop aggregate signature scheme, and how to use a hashed multi-hop seBARG to construct an IVC scheme.

## 6.1 Definition

To formally capture the notion of multi-hop seBARGs, we make the following syntactic and semantic changes to seBARGs, which includes adding a new “proof combiner” procedure that we call **AggProve**.

**Syntax.** A rate-1 multi-hop seBARG scheme for an NP language  $\mathcal{L}$  consists of the following polynomial time algorithms:

$\text{Gen}(1^\lambda, d, (i_1, \dots, i_d)) \rightarrow (\text{crs}, \text{td})$ . This is a probabilistic setup algorithm. It no longer takes as input the number of instances  $k$  being batched, nor the length of the instances  $n$ .<sup>13</sup> Rather, it takes as input the security parameter  $1^\lambda$ , the maximum number of hops  $d \in [2^\lambda]$  (i.e., the number of batch-compositions), and a sequence of  $d$  extraction indices  $I = (i_1, \dots, i_d) \in [2^\lambda]^d$ . It outputs  $\text{crs}$  which consists of  $d$  strings  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_d)$ , along with a trapdoor  $\text{td}$  which consists of  $d$  strings  $\text{td} = (\text{td}_1, \dots, \text{td}_d)$ .

We note that in the multi-hop setting, we need to define a separate extraction index for each hop. That is, instead of having a single extraction index (as in a standard seBARG), it takes as input  $d$  extraction indices  $I = (i_1, \dots, i_d)$ . Also, since the batch size in each hop is unbounded, each index  $i_j$  is an element in  $[2^\lambda]$ . The  $j$ 'th extraction index  $i_j$  is interpreted as saying that, from an accepting seBARG proof  $\pi$  created via  $j$ -hops/compositions, we can efficiently extract the  $i_j$ 'th witness (which is another accepting seBARG proof  $\pi'$  created via  $(j - 1)$ -hops/compositions). Thus, the sequence of extraction indices  $I$  defines the sequence in which multi-hop seBARGs can be recursively extracted.

$\mathcal{P}(\text{crs}_1, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \rightarrow \pi$ . The prover  $\mathcal{P}$  is a poly-time algorithm that takes as input  $\text{crs}_1$  (which is the first string in  $\text{crs}$ ), any (unbounded) number of instance-witness pairs of arbitrary size (since the batch size nor the instance size are no longer fixed at setup time). It runs in time  $\text{poly}(\lambda, (|x_i|, |w_i|)_{i \in [k]})$  and outputs a proof  $\pi$ .

$\text{AggProve}((\text{crs}_i)_{i \in [d]}, X^{(1)}, \dots, X^{(\ell)}, \pi^{(1)}, \dots, \pi^{(\ell)}) \rightarrow \pi$ . This proof combiner is a poly-time algorithm that takes as input the  $(\text{crs}_i)_{i \in [d]}$ , a sequence of arbitrarily many *instance-trees*  $X^{(1)}, \dots, X^{(\ell)}$  (as defined below), of maximal depth  $d' - 1$ , and corresponding multi-BARG proofs  $\pi^{(1)}, \dots, \pi^{(\ell)}$ . It outputs a (combined) multi-BARG proof  $\pi$ .

**Definition 6.1.** *An instance-tree  $X$  is a tree of varying arity, where each leaf node  $v$  is associated with an instance  $x_v$ , and each intermediate node  $u$  corresponds to a multi-BARG proof that certifies the validity of the sub-tree rooted at  $u$  (these multi-BARG proofs are not included in  $X$ ).*

We note that the depth of an instance-tree  $X$  represents the number of hops taken in order to compute a proof  $\pi$  for  $X$ . Therefore, the proof combiner takes as input a sequence of instance-trees along with their corresponding multi-BARG proofs, where each instance-tree encodes not only the instances in the leaves, but also the history of how the previous proofs were combined, including the number of hops taken so far.

<sup>13</sup>We note that it was unnecessary for **Gen** to take  $k$  and  $n$  as input to begin with, since it could have produced a  $\text{crs}$  corresponding to every  $k, n \in \{2^i\}_{i \in [\lambda]}$ , and by padding we can assume w.l.o.g. that indeed  $k, n$  are powers of 2. Indeed, originally **Gen** took  $k, n$  as input only for the sake of simplicity.



$\mathcal{V}((\text{crs}_i)_{i \in [d]}, X, \pi) \rightarrow \{0, 1\}$ . The verifier  $\mathcal{V}$  is a poly-time algorithm that takes as input  $(\text{crs}_i)_{i \in [d]}$ , an instance-tree  $X$  of depth  $d'$ , and a (combined) proof  $\pi$ , and outputs 0/1 (corresponding to reject or accept).

**Notation 6.2.** *In what follows, when we refer to a tree  $T$  we always think of a tree where each of its leaves, denoted by  $v$ , is associated with a parameter  $n_v = n_v(\lambda)$ .<sup>14</sup> When we refer to a poly-size tree, we mean that the number of nodes in the tree is  $\leq \text{poly}(\lambda)$  and  $n_v = n_v(\lambda) \leq \text{poly}(\lambda)$  for every leaf  $v$ . For any tree  $T$ , we let  $\text{path}(T)$  be the set that consists of all the possible paths from the root to a leaf in  $T$ . We say that an instance-tree  $X$  is consistent with  $T$  if  $X$  has the exact same tree structure as  $T$  and for each leaf  $v \in T$  the instance  $x_v$  in  $X$  (corresponding to the leaf  $v$ ) is of size  $n_v$ . For any instance-tree  $X$  that is consistent with  $T$  and any  $(i_1, \dots, i_d) \in \text{path}(T)$  we let  $X_{i_1, \dots, i_d}$  denote the instance in the leaf corresponding to the path  $(i_1, \dots, i_d)$ .*

*In addition, for any instance-trees  $X^{(1)}, \dots, X^{(\ell)}$  we denote by  $X = (X^{(1)}, \dots, X^{(\ell)})$  the instance-tree that combines all the  $\ell$  instance-trees  $X^{(1)}, \dots, X^{(\ell)}$  by adding a root with arity  $\ell$ , whose  $i$ 'th child is the root of  $X^{(i)}$ .*

**Definition 6.3.** *A rate-1 multi-hop seBARG scheme  $\text{multi-BARG} = (\text{Gen}, \mathcal{P}, \text{AggProve}, \mathcal{V})$  for an NP language  $\mathcal{L}$  is required to satisfy the following properties (the same properties in definitions 3.3 and 3.6, adapted to multi-hop syntax, with an additional completeness and compactness guarantee for combined proofs):*

**Efficiency.** *For every  $i \in [d]$ , the size of  $(\text{crs}_i, \text{td}_i)$  is at most  $\text{poly}(\lambda)$ , and the size of a (combined) proof corresponding to an instance-tree  $X$  of depth  $d'$  is at most  $m + d' \cdot \text{poly}(\lambda, \log|X|)$ , where  $m$  is the maximal witness length of all the leaf instances in  $X$ .*

**Completeness.** *For any  $\lambda \in \mathbb{N}$ , any  $d \in [2^\lambda]$ , any instance-tree  $X$  of size  $\leq 2^\lambda$  and depth  $d' \leq d$ , and any corresponding valid witness  $W$ ,*

$$\Pr \left[ \begin{array}{l} \mathcal{V}((\text{crs}_i)_{i \in [d]}, X, \pi) = 1 \\ \text{parse crs} = (\text{crs}_i)_{i \in [d]}, \\ \pi \leftarrow \text{Agg}((\text{crs}_i)_{i \in [d]}, X, W) \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, d, (i_1, \dots, i_d)), \\ \end{array} \right] = 1$$

where  $\text{Agg}((\text{crs}_i)_{i \in [d]}, X, W)$  is defined by induction on  $d'$  as follows:

If  $d' = 1$  then parse  $X = (x_1, \dots, x_\ell)$  and  $W = (w_1, \dots, w_\ell)$  and output

$$\text{Agg}(\text{crs}_1, X, W) = \mathcal{P}(\text{crs}_1, x_1, \dots, x_\ell, w_1, \dots, w_\ell).$$

If  $d' > 1$  then parse  $X = (X^{(1)}, \dots, X^{(\ell)})$  and  $W = (W^{(1)}, \dots, W^{(\ell)})$ , where  $W^{(i)}$  is the witness corresponding to the sub-tree instance  $X^{(i)}$ . Denote by  $d_i$  the depth of  $X^{(i)}$ .

For every  $i \in [\ell]$  compute by induction

$$\pi^{(i)} = \text{Agg}((\text{crs}_j)_{j \in [d_i]}, X^{(i)}, W^{(i)})$$

and output

$$\pi = \text{AggProve}((\text{crs}_i)_{i \in [d]}, X^{(1)}, \dots, X^{(\ell)}, \pi^{(1)}, \dots, \pi^{(\ell)}).$$

<sup>14</sup>Different leaves can be associated with different parameters.

**Index hiding.** For any poly-size adversary  $\mathcal{A}$ , any polynomial  $d = \text{poly}(\lambda)$ , any poly-size tree  $T$  of depth  $d' \leq d$ , and any sets of indices  $I_0 = (i_{0,1}, \dots, i_{0,d}), I_1 = (i_{1,1}, \dots, i_{1,d}) \in \text{path}(T)$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

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

**Somewhere argument of knowledge.** There exists a PPT extractor  $\mathcal{E}$  such that for any poly-size adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for any polynomial  $d = \text{poly}(\lambda)$ , any poly-size tree  $T$  of depth  $d' \leq d$ , and any set of indices  $(i_1, \dots, i_{d'}) \in \text{path}(T)$  and  $i_{d'+1}, \dots, i_d \in [2^\lambda]$ , for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, X, \pi) = 1 \\ \wedge X \text{ is consistent with } T \\ \wedge w^* \text{ is not a valid witness for } X_{i_1, \dots, i_{d'}} \in \mathcal{L} \end{array} \quad : \quad \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, d, (i_1, \dots, i_d)) \\ (X, \pi) = \mathcal{A}(\text{crs}) \\ w^* \leftarrow \mathcal{E}(\text{td}, X, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

**Remark 6.1.** We note that the above somewhere argument of knowledge property implies the following semi-adaptive soundness property:

For any poly-size adversary  $\mathcal{A}$ , any polynomial  $d = \text{poly}(\lambda)$ , any poly-size tree  $T$  of depth  $d' \leq d$ , and any set of indices  $(i_1, \dots, i_{d'}) \in \text{path}(T)$  and  $i_{d'+1}, \dots, i_d \in [2^\lambda]$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, X, \pi) = 1 \\ \wedge X \text{ is consistent with } T \\ \wedge X_{i_1, \dots, i_{d'}} \notin \mathcal{L} \end{array} \quad : \quad \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, d, (i_1, \dots, i_d)) \\ (X, \pi) = \mathcal{A}(\text{crs}) \end{array} \right] \leq \text{negl}(\lambda).$$

## 6.2 Construction and Analysis

In this section we construct a multi-hop seBARG from any rate-1 seBARG (Definitions 3.3 and 3.6). We do this in two steps. First, we show how to convert the rate-1 seBARG into an intermediate primitive that we call a rate-1 *single-hop* seBARG, denoted by *single-seBARG*, and then we show how to convert a rate-1 *single-seBARG* into a *multi-hop* seBARG.

**Single-hop seBARG.** A *single-seBARG* scheme is very similar to a seBARG scheme, the only difference being that it enables batching instances and witnesses of varying lengths, and these lengths, as well as batch size  $k$ , can be decided at batching time (and are no longer fixed at setup time). The reason why a rate-1 *single-seBARG* can be directly obtained from a rate-1 seBARG scheme follows from the following two observations about a rate-1 seBARG scheme.

**Observation 1.** The fact that the running time of a rate-1 seBARG.Gen grows only *poly-logarithmically* with the number of instances  $k$  and the input length  $n$  (see Definition 3.6) implies that one could generically convert this seBARG into another seBARG scheme in which the number of instances and the input length are not a priori bounded. To see this, we first observe that by standard padding we can consider only  $k$  and  $n$  which are powers of 2. Therefore, we can run seBARG.Gen  $\lambda^2$  times, one for each  $k, n \in \{2^i\}_{i \in [\lambda]}$ . The prover can then compute the seBARG proof using the crs corresponding to appropriate  $k, n$ . This gives a *single-hop seBARG* scheme for  $k$  and  $n$  that are not fixed in advance.

**Observation 2.** Our second observation is that any rate-1 seBARG can be converted into one that allows batching instances of unequal lengths, by padding all the instances and witnesses to be of the same length.

By combining the above two observations we can convert our rate-1 seBARG scheme into a rate-1 single-seBARG scheme, which enables batching instances of varying lengths where the length of these instances as well as the batch size are not a priori fixed.

**The construction of our rate-1 multi-hop seBARG.** Fix any NP language  $\mathcal{L}$ . We construct a rate-1 multi-hop seBARG scheme for  $\mathcal{L}$ , denoted by multi-BARG, which uses as a building block a rate-1 single-seBARG scheme for the NP language  $\mathcal{L}'$ . This NP language  $\mathcal{L}'$  contains  $\mathcal{L}$  and is defined recursively below. Loosely speaking, any instance in  $\mathcal{L}'$  is an instance-tree  $X$  of some depth  $d'$ , along with  $(\text{crs}_i)_{i \in [d']}$  (if  $d' = 0$  then there is no crs associated with the instance) and a valid witness is a valid aggregated proof corresponding to  $(\text{crs}_i)_{i \in [d]}$  (where if  $d' = 0$  then a valid proof is simply a valid witness corresponding to  $\mathcal{L}$ ).

$\text{Gen}(1^\lambda, d, I) \rightarrow (\text{crs}, \text{td})$ . This PPT algorithm parses  $I = (i_1, \dots, i_d)$ , it samples for every  $j \in [d]$

$$(\text{crs}_j, \text{td}_j) \leftarrow \text{single-seBARG.Gen}(1^\lambda, i_j),$$

and it outputs  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_d)$  and  $\text{td} = (\text{td}_1, \dots, \text{td}_d)$ .

$\mathcal{P}(\text{crs}_1, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \rightarrow \pi$ . This poly-time algorithm outputs

$$\pi \leftarrow \text{single-seBARG.P}(\text{crs}_1, x_1, \dots, x_k, \omega_1, \dots, \omega_k).$$

$\text{AggProve}(\text{crs}, X^{(1)}, \dots, X^{(\ell)}, \pi^{(1)}, \dots, \pi^{(\ell)}) \rightarrow \pi$ . For every  $i \in [\ell]$  denote by  $d_i$  the depth of  $X^{(i)}$  and let  $d' = \max\{d_i\} + 1$ . If  $d' > d$  then abort. Otherwise, output

$$\pi \leftarrow \text{single-seBARG.P} \left( \text{crs}_{d'}, \left( (\text{crs}_j)_{j \in [d_i]}, X^{(i)} \right)_{i \in [\ell]}, \left( \pi^{(i)} \right)_{i \in [\ell]} \right)$$

where a valid witness corresponding to  $((\text{crs}_j)_{j \in [d_i]}, X^{(i)}) \in \mathcal{L}'$  is  $\pi^{(i)}$  such that

$$\mathcal{V} \left( (\text{crs}_j)_{j \in [d_i]}, X^{(i)}, \pi^{(i)} \right) = 1$$

where if  $d_i = 0$  then we define  $\mathcal{V}(X^{(i)}, \pi^{(i)}) = 1$  if and only if  $\pi^{(i)}$  is a valid witness for  $X^{(i)} \in \mathcal{L}$ .

$\mathcal{V}(\text{crs}, X, \pi) \rightarrow 0/1$ . Parse  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_{d'})$  and parse  $X = (X^{(1)}, \dots, X^{(\ell)})$ . Let  $d_i \in [d]$  be the depth of the instance-tree  $X^{(i)}$ . If  $d' \neq \max\{d_i\}_{i \in [\ell]} + 1$  then output 0. Else, output

$$\text{single-seBARG.V} \left( \text{crs}_{d'}, \left( (\text{crs}_j)_{j \in [d_i]}, X^{(i)} \right)_{i \in [\ell]}, \pi \right).$$

**Remark 6.2** (The language  $\mathcal{L}'$ ). At first there may appear to be a circularity in the definition of the NP language  $\mathcal{L}'$ , since on the one hand its definition depends on the multi-hop multi-BARG scheme, and on the other hand, multi-BARG uses as a building block a rate-1 single-seBARG

for  $\mathcal{L}'$ . The reason this is not an issue is that both  $\mathcal{L}'$  and multi-BARG are formally defined by induction on the depth  $d'$  of the instance-trees, starting with  $d' = 0$ . Namely, we define by induction for every  $d' \in \mathbb{N}$ ,  $\mathcal{L}'_{d'}$  and  $\mathcal{V}_{d'}$ , where  $\mathcal{L}'_{d'}$  contains only instance-trees of depth at most  $d'$  (where  $\mathcal{L}'_0 = \mathcal{L}$ ), and  $\mathcal{V}_{d'}$  takes as input only an instance-tree of depth  $d'$ , and it runs `single-seBARG.V` with instance-trees of depth  $d' - 1$  (where  $\mathcal{L}'_{d'-1}$  is already defined by induction).

**Theorem 6.4.** *If single-seBARG is a rate-1 single-hop seBARG scheme then multi-BARG is a rate-1 multi-hop seBARG scheme (definition 6.3).*

*Proof.* In what follows we prove that multi-BARG satisfies all the desired properties:

**Efficiency.** Follows immediately from the fact that the underlying single-seBARG is rate-1.

**Completeness.** Follows directly from the completeness of the underlying single-seBARG scheme.

**Index-hiding.** Follows immediately from the index-hiding property of the underlying single-seBARG scheme (via a standard hybrid argument).

**Somewhere argument of knowledge.** Let `single-seBARG.E` denote the PPT extractor corresponding to the underlying single-seBARG scheme. We define an extractor  $\mathcal{E}$  that given  $(\mathbf{td}, X, \pi)$  does the following:

1. Parse  $\mathbf{td} = (\mathbf{td}_1, \dots, \mathbf{td}_d)$  and suppose that  $\mathbf{td}$  corresponds to indices  $(i_1, \dots, i_d)$ .
2. Parse  $X = (X^{(1)}, \dots, X^{(\ell)})$ , and let  $d'$  denote the depth of  $X$ .
3. For every  $j \in [d']$  we denote by  $d_j$  the depth of the subtree  $X^{(i_{d'}, \dots, i_{d'+1-j})}$ .

Let  $j^* \in [d']$  be the smallest index such that  $d_j = 0$  for every  $j \geq j^*$  (i.e.,  $j^*$  is the length of the path  $(i_{d'}, \dots, i_1)$  in  $X$  until we reach a leaf).

4. Compute

$$\pi^{(i_{d'})} = \text{single-seBARG.E}(\mathbf{td}_{d'}, X, \pi).$$

Intuitively, if  $\pi$  is a valid multi-BARG proof for  $X$  then with overwhelming probability  $\pi^{(i_{d'})}$  is a valid multi-BARG proof for the instance-tree  $X^{(i_{d'})}$ .

5. For every  $j \in [j^* - 1]$  we compute (by induction, starting with  $j = 1$ )

$$\pi^{(i_{d'}, \dots, i_{d'-j})} = \text{single-seBARG.E} \left( \mathbf{td}_{d_j}, X^{(i_{d'}, \dots, i_{d'-j+1})}, \pi^{(i_{d'}, \dots, i_{d'-j+1})} \right)$$

where  $X^{(i_{d'}, \dots, i_{d'-j})}$  is defined to be the subtree of  $X$  obtained by going down from the root on the path  $(i_{d'}, \dots, i_{d'-j})$ .

6. Output  $\omega = \pi^{(i_{d'}, \dots, i_{d'-j^*+1})}$ .

Fix any poly-size adversary  $\mathcal{A}$  and any polynomial  $d = \text{poly}(\lambda)$ . In what follows we argue by induction that for every  $d' \leq d$  there exists a negligible function  $\mu_{d'}$ , such that for any poly-size tree  $T$  of depth  $d'$ , and any set of indices  $(i_1, \dots, i_{d'}) \in \text{path}(T)$  and  $i_{d'+1}, \dots, i_d \in [2^\lambda]$  it holds that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}((\text{crs}_j)_{j \in [d']}, X, \pi) = 1 \\ \wedge X \text{ is consistent with } T \\ \wedge w^* \text{ is not a valid witness for } X_{i_1, \dots, i_{d'}} \in \mathcal{L} \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, d, (i_1, \dots, i_d)) \\ (X, \pi) = \mathcal{A}(\text{crs}) \\ w^* \leftarrow \mathcal{E}(\text{td}, X, \pi) \end{array} \right] \leq \mu_{d'}(\lambda). \quad (14)$$

**Base case:**  $d' = 1$ : This follows immediately from the somewhere argument of knowledge of the underlying single-seBARG scheme.

**Induction step:** Suppose that Equation (14) holds for every  $j < d'$  and we prove that it holds for  $d'$ . To this end, the somewhere argument of knowledge of the underlying single-seBARG scheme implies that there exists a negligible function  $\nu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}((\text{crs}_j)_{j \in [d']}, X, \pi) = 1 \\ \wedge X \text{ is consistent with } T \\ \wedge \mathcal{V}((\text{crs}_j)_{j \in [d_1]}, X^{(i_{d'})}, \pi^{(i_{d'})}) = 1 \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, d, (i_1, \dots, i_d)) \\ (X, \pi) = \mathcal{A}(\text{crs}) \\ \pi^{(i_{d'})} \leftarrow \text{single-seBARG.}\mathcal{E}(\text{td}_{d'}, X, \pi) \\ \text{parse crs} = (\text{crs}_1, \dots, \text{crs}_d) \end{array} \right] \leq \nu(\lambda). \quad (15)$$

Consider the poly-size adversary  $\mathcal{A}'$  that given  $\text{crs}$  does the following:

1. Parse  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_d)$ .
2. Generate  $(\text{crs}^*, \text{td}^*) \leftarrow \text{single-seBARG.Gen}(1^\lambda, i_{d'})$ .
3. Let  $\text{crs}^* = (\text{crs}_1, \dots, \text{crs}_{d'-1}, \text{crs}^*, \text{crs}_{d'+1}, \dots, \text{crs}_d)$ .
4. Compute  $(X, \pi) = \mathcal{A}(\text{crs}^*)$ .
5. Compute  $\pi^{(i_{d'})} = \text{single-seBARG.}\mathcal{E}(\text{td}^*, X, \pi)$ .
6. Output  $(X^{(i_{d'})}, \pi^{(i_{d'})})$ .

By the induction hypothesis there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}((\text{crs}_j)_{j \in [d_1]}, X^{(i_{d'})}, \pi^{(i_{d'})}) = 1 \\ \wedge X^{(i_{d'})} \text{ is consistent with } T^{(i_{d'})} \\ \wedge \mathcal{V}((\text{crs}_j)_{j \in [d_2]}, X^{(i_{d'}, i_{d'-1})}, \pi^{(i_{d'}, i_{d'-1})}) = 1 \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, d, (i_1, \dots, i_d)) \\ (X^{(i_{d'})}, \pi^{(i_{d'})}) = \mathcal{A}'(\text{crs}) \\ \pi^{(i_{d'}, i_{d'-1})} \leftarrow \mathcal{E}(\text{td}^*, X^{(i_{d'})}, \pi^{(i_{d'})}) \end{array} \right] \leq \nu(\lambda).$$

This, together with Equation (15), implies the induction step (i.e., Equation (14)). □

## 6.3 Hashed Multi-Hop seBARGs

### 6.3.1 Definition

A hashed multi-BARG is one where the verifier is required to verify the proof without being given the instances (in the clear), and instead being given only a hash of the instances. In Section 7 we use our hashed multi-BARG to construct an IVC scheme. For this application, it suffices to consider hashed multi-BARG where all the instances are of the same size  $n$  and where the number of instances being BARGed is a fixed parameter  $k$ . Therefore, for simplicity we fix  $n$  and  $k$  in setup time. We also consider a hashed multi-BARG where the number of hops is at most  $\lambda$ . This suffices for our IVC application since there we will combine the seBARGs in a tree-like structure with fan-out  $k$ , so a depth  $\lambda$  tree is sufficient to accumulate  $k^\lambda$  many proofs.

We note that one could define the more general notion where  $n$  and  $k$  can vary adaptively (as in Section 6.1), though we chose simplicity over generality in this section.

**Syntax.** A hashed multi-BARG scheme corresponding to an NP language  $\mathcal{L}$  is associated with the following algorithms:

$\text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)) \rightarrow (\text{crs}, \text{td})$ . This is a PPT algorithm that takes as input a security parameter  $1^\lambda$ , an instance size  $n$ , the number of instances  $k$ , and a sequence of  $\lambda$  extraction indices  $i_1, \dots, i_\lambda \in [k]$ . It outputs a  $\text{crs}$  and a trapdoor key  $\text{td}$ .

$\mathcal{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \rightarrow (\mathbf{v}, \pi)$ . This is a poly-time prover algorithm that takes as input  $\text{crs}$ , instances  $x_1, \dots, x_k$  each of size  $n$ , and corresponding witnesses  $\omega_1, \dots, \omega_k$ , and outputs a hash value  $\mathbf{v}$  and a proof  $\pi$ .

$\text{AggProve}(\text{crs}, \mathbf{v}^{(1)}, \dots, \mathbf{v}^{(k)}, \pi^{(1)}, \dots, \pi^{(k)}) \rightarrow (\mathbf{v}, \pi)$ . This proof combiner is a poly-time algorithm that takes as input  $\text{crs}$ , a sequence of hash values  $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(k)}$ , along with corresponding proofs  $\pi^{(1)}, \dots, \pi^{(k)}$ , and outputs a new hash value  $\mathbf{v}$  and a corresponding proof  $\pi$ .

$\mathcal{V}(\text{crs}, \mathbf{v}, \pi) \rightarrow 0/1$ . The verifier  $\mathcal{V}$  is a poly-time algorithm that takes as input  $\text{crs}$ , a hash value  $\mathbf{v}$  and a proof  $\pi$  and outputs a bit to signal whether the proof is valid or not.

$\text{Extract}(\text{td}, \mathbf{v}, I) \rightarrow x$ . This is a poly-time extraction algorithm that takes as input a trapdoor  $\text{td}$ , a hash value  $\mathbf{v}$ , and a sequence of indices  $I = (i_1, \dots, i_\lambda) \in [k]^\lambda$  and outputs an instance  $x$ .

**Remark 6.3.** The fact that in a hashed multi-BARG the parameters  $n$  and  $k$  are fixed in advance, implies that the instance tree is a  $k$ -ary tree where all the leaves correspond to instances of size  $n$ . Since for any number of instances  $\ell$  that are being combined, this tree is fixed, we do not need to include it as an input; rather it suffices to include only the number of instances  $\ell$ , which we assume w.l.o.g. is encoded in  $\mathbf{v}$ . We note that when defining a (regular) multi-BARG in Section 6.1 we did need to include it as part of the input since  $n$  and  $k$  were adaptively chosen and were changing from hop to hop.

In what follows we define the properties of a hashed multi-BARG. In the definition we assume that the number of instances  $\ell$  is a power of  $k$ . This simplifies the definition significantly and is sufficient for the following reason: In our construction, when we aggregate an arbitrary number of instances  $\ell \in [2^\lambda]$ , letting  $\ell = \sum_{i=0}^{\lambda} c_i k^i$  where  $c_i \in \{0, 1, \dots, k-1\}$ , our aggregated proof will consist of  $\sum_{i=0}^{\lambda} c_i$  aggregated proofs, where for every  $i \in \{0, 1, \dots, \lambda\}$  we have  $c_i$  aggregated proofs for  $k^i$  instances.

**Definition 6.5.** A rate-1 hashed multi-BARG for an NP language  $\mathcal{L}$  is required to satisfy the following properties.

**Efficiency.** The output  $(v, \pi)$  generated by the prover, or the aggregate prover, is of size at most  $n \cdot \text{poly}(\lambda) + m$  where  $n$  is the instance size and  $m$  is the witness size.<sup>15</sup>

**Completeness.** For any  $\lambda \in \mathbb{N}$  any  $n, k, \ell \in [2^\lambda]$  such that  $k \geq 2$  and  $\ell = k^d$  for some  $d \in \mathbb{N}$ , any  $\ell$  instances  $x_1, \dots, x_\ell$  each of size  $n$ , and corresponding valid witnesses  $\omega_1, \dots, \omega_\ell$ , and any  $i_1, \dots, i_\lambda \in [k]$ ,

$$\Pr \left[ \mathcal{V}(\text{crs}, v, \pi) = 1 \quad : \quad \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)), \\ (v, \pi) \leftarrow \text{Agg}(\text{crs}, x_1, \dots, x_\ell, \omega_1, \dots, \omega_\ell) \end{array} \right] = 1$$

where  $\text{Agg}(\text{crs}, x_1, \dots, x_\ell, \omega_1, \dots, \omega_\ell)$  is defined as follows: If  $\ell = k$  then

$$\text{Agg}(\text{crs}, x_1, \dots, x_\ell, \omega_1, \dots, \omega_\ell) \triangleq \mathcal{P}(\text{crs}, x_1, \dots, x_\ell, \omega_1, \dots, \omega_\ell).$$

Otherwise, for  $i \in [k]$  inductively compute

$$\left( v^{(i)}, \pi^{(i)} \right) = \text{Agg} \left( \text{crs}, x_{(i-1) \cdot \frac{\ell}{k} + 1}, \dots, x_{i \cdot \frac{\ell}{k}}, \omega_{(i-1) \cdot \frac{\ell}{k} + 1}, \dots, \omega_{i \cdot \frac{\ell}{k}} \right)$$

and output

$$\text{AggProve} \left( \text{crs}, v^{(1)}, \dots, v^{(k)}, \pi^{(1)}, \dots, \pi^{(k)} \right).$$

**Index hiding.** For any poly-size adversary  $\mathcal{A}$ , any polynomials  $n = n(\lambda)$  and  $k = k(\lambda)$ , and any sets of indices  $I_0 = I_0(\lambda) = (i_{0,1}, \dots, i_{0,\lambda}), I_1 = I_1(\lambda) = (i_{1,1}, \dots, i_{1,\lambda}) \in [k]^\lambda$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

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

**Extraction correctness.** For any  $\lambda \in \mathbb{N}$  any  $n, k, \ell \in [2^\lambda]$  such that  $k \geq 2$  and  $\ell = k^d$  for some  $d \in \mathbb{N}$ , any  $\ell$  instances  $x_1, \dots, x_\ell$  each of size  $n$ , and corresponding valid witnesses  $\omega_1, \dots, \omega_\ell$ , any  $i \in [\ell]$ , letting the  $k$ -ary representation of  $i$  be  $i = (i_1, \dots, i_d)$ , for any  $i_{d+1}, \dots, i_\lambda \in [k]$ ,

$$\Pr \left[ x_i \neq \text{Extract}(\text{td}, v, I) \quad : \quad \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)), \\ (v, \pi) \leftarrow \text{AggProve}(\text{crs}, x_1, \dots, x_\ell, \omega_1, \dots, \omega_\ell) \end{array} \right] \leq \text{negl}(\lambda).$$

**Somewhere argument of knowledge.** There exists a PPT extractor  $\mathcal{E}$  such that for any poly-size adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for any polynomials  $n = n(\lambda)$  and  $k = k(\lambda)$ , and any sequence of indices  $I = I(\lambda) = (i_1, \dots, i_\lambda) \in [k]^\lambda$ , for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, \text{hk}, v, \pi) = 1 \\ \wedge \left( x \neq \text{Extract}(\text{td}, v, I) \right) \\ \vee w \text{ is not a valid witness for } x \in \mathcal{L} \end{array} \quad : \quad \begin{array}{l} (\text{crs}, \text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, n, k, I) \\ (v, \pi) = \mathcal{A}(\text{crs}, \text{hk}) \\ (x, w) \leftarrow \mathcal{E}(\text{td}, v, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

<sup>15</sup>Our scheme allows for at most  $\lambda$  hops so we do not need to incorporate the depth as an additional parameter.

### 6.3.2 Construction and Analysis

Fix any NP language  $\mathcal{L}$  and any rate-1 fully-local somewhere extractable hash family

$$\text{SEH} = (\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify}, \text{SEH.Extract}).$$

We construct a hashed multi-BARG scheme for  $\mathcal{L}$  w.r.t. the SEH family. Our scheme uses as a building block a rate-1 seBARG scheme

$$\text{seBARG} = (\text{seBARG.Gen}, \text{seBARG.P}, \text{seBARG.V}, \text{seBARG.Extract})$$

for the NP language  $\mathcal{L}'$  defined recursively below.<sup>16</sup>

$\text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)) \rightarrow (\text{crs}, \text{td})$ . This PPT algorithm does the following:

1. For every  $d \in [\lambda]$  sample  $(\text{hk}_d, \text{SEH.td}_d) \leftarrow \text{SEH.Gen}(1^\lambda, n_{d-1} \cdot k, I_{i_d})$ , where  $n_0 = n$  and for every  $d \in [\lambda]$ ,  $n_d$  is the size of hash values outputted by  $\text{SEH.Hash}(\text{hk}_d, \cdot)$ , i.e., hash values of depth  $d$ , and

$$I_{i_d} \triangleq \{n_{d-1} \cdot (i_d - 1) + 1, \dots, n_{d-1} \cdot i_d\}.$$

Note that  $n_d \leq n_{d-1} + \text{poly}(\lambda)$  and thus  $n_\lambda \leq n + \text{poly}(\lambda)$ .

2. For every  $d \in [\lambda]$  sample  $(\text{crs}_d, \text{td}_d) \leftarrow \text{seBARG.Gen}(1^\lambda, n'_d, k, i_d)$ , where  $n'_d$  for  $d \in [\lambda]$  is defined below.
3. Let  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_\lambda, \text{hk}_1, \dots, \text{hk}_\lambda)$ , and  $\text{td} = (\text{td}_1, \dots, \text{td}_\lambda, \text{SEH.td}_1, \dots, \text{SEH.td}_\lambda)$ .
4. Output  $(\text{crs}, \text{td})$ .

$\mathcal{P}(\text{crs}, x_1, \dots, x_k, \omega_1, \dots, \omega_k) \rightarrow (\mathbf{v}, \pi)$ . This poly-time algorithm does the following:

1. Parse  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_\lambda, \text{hk}_1, \dots, \text{hk}_\lambda)$ .
2. Compute  $\mathbf{v} = \text{SEH.Hash}(\text{hk}_1, (x_1, \dots, x_k))$ .  
Note that  $|\mathbf{v}| \leq n + \text{poly}(\lambda)$ .
3. For every  $i \in [k]$  compute  $\rho_i = \text{SEH.Open}(\text{hk}_1, (x_1, \dots, x_k), I_i)$ , where  $I_i$  corresponds to the coordinates of  $x_i$ , i.e.,

$$I_i \triangleq \{n \cdot (i - 1) + 1, \dots, n \cdot i\}.$$

Note that  $|\rho_i| \leq \text{poly}(\lambda)$  (see Lemma 3.2 in Section 3.1).

4. Compute  $\pi = \text{seBARG.P}(\text{crs}_1, (\text{hk}_1, \mathbf{v}, i)_{i \in [k]}, (x_i, \rho_i, w_i)_{i \in [k]})$ , where instances of  $\mathcal{L}'$  have length

$$n'_1 = |(\text{hk}_1, \mathbf{v}, i)| \leq n + \text{poly}(\lambda)$$

and a valid witness  $(x_i, \rho_i, w_i)$  satisfies the following conditions:

$$\text{SEH.Verify}(\text{hk}_1, \mathbf{v}, I_i, x_i, \rho_i) = 1 \quad \wedge \quad (x_i, w_i) \in \mathcal{R}_{\mathcal{L}}.$$

The fact that seBARG and SEH both have rate 1 implies that  $\pi$  is of size

$$|\pi| \leq |(x_i, \rho_i, w_i)| + \text{poly}(\lambda) \leq n + m + \text{poly}(\lambda),$$

as desired.

---

<sup>16</sup>The language  $\mathcal{L}'$  is somewhat similar to the NP language  $\mathcal{L}'$  defined in the construction of the multi-BARG scheme in Section 6.2, except that here the instance-tree is hashed.



5. Output  $(\mathbf{v}, \pi)$ .

$\text{AggProve}(\text{crs}, \mathbf{v}^{(1)}, \dots, \mathbf{v}^{(k)}, \pi^{(1)}, \dots, \pi^{(k)}) \rightarrow (\mathbf{v}, \pi)$ . This poly-time algorithm does the following:

1. Parse  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_\lambda, \text{hk}_1, \dots, \text{hk}_\lambda)$ .
2. Check that  $d(\mathbf{v}^{(i)}) = d(\mathbf{v}^{(j)})$  for every  $i, j \in [k]$  and that this depth, denoted by  $d$ , is smaller than  $\lambda$ . If this is not the case then abort.
3. Compute  $\mathbf{v} = \text{SEH.Hash}(\text{hk}_{d+1}, (\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(k)}))$ . Note that

$$|\mathbf{v}| \leq |\mathbf{v}^{(i)}| + \text{poly}(\lambda) = n_d + \text{poly}(\lambda) \leq n + \text{poly}(\lambda).$$

4. For every  $i \in [k]$  compute  $\rho_i = \text{SEH.Open}(\text{hk}_{d+1}, (\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(k)}), I_i)$ , where (as above)  $I_i$  corresponds to the coordinates of  $\mathbf{v}^{(i)}$ ; i.e.,

$$I_i \triangleq \{n_d \cdot (i - 1) + 1, \dots, n_d \cdot i\}$$

and  $|\rho_i| \leq \text{poly}(\lambda)$ .

5. Compute

$$\pi = \text{seBARG.P} \left( \text{crs}_{d+1}, \begin{array}{l} \text{instances} = ((\text{crs}_j)_{j \in [d]}, (\text{hk}_j)_{j \in [d+1]}, \mathbf{v}, i)_{i \in [k]} \\ \text{witnesses} = (\mathbf{v}^{(i)}, \rho_i, \pi^{(i)})_{i \in [k]} \end{array} \right),$$

where instances of  $\mathcal{L}'$  have length

$$n'_{d+1} = |(\text{crs}_j)_{j \in [d]}, (\text{hk}_j)_{j \in [d+1]}, \mathbf{v}, i| \leq n + \text{poly}(\lambda)$$

and a valid witness  $(\mathbf{v}^{(i)}, \rho_i, \pi^{(i)})$  satisfies the following conditions:

(a)  $\rho_i$  is a valid opening of  $\mathbf{v}^{(i)}$  w.r.t.  $(\text{hk}_{d+1}, \mathbf{v})$ . Namely,

$$\text{SEH.Verify}(\text{hk}_{d+1}, \mathbf{v}, I_i, \mathbf{v}^{(i)}, \rho_i) = 1$$

(b)  $\pi^{(i)}$  is a valid proof w.r.t.  $\mathbf{v}^{(i)}$ . Namely,

$$\text{seBARG.V} \left( \text{crs}_d, \left( (\text{crs}_j)_{j \in [d-1]}, (\text{hk}_j)_{j \in [d]}, \mathbf{v}^{(i)}, j \right)_{j \in [k]}, \pi^{(i)} \right) = 1.$$

Inductively we have that  $|\pi^{(i)}| \leq n \cdot d \cdot \text{poly}(\lambda) + m$ . The fact that seBARG and SEH both have rate 1 implies that  $\pi$  is of size

$$\begin{aligned} |\pi| &= \left| (\mathbf{v}^{(i)}, \rho_i, \pi^{(i)}) \right| + \text{poly}(\lambda) \leq n_d + \text{poly}(\lambda) + (n \cdot d \cdot \text{poly}(\lambda) + m) \\ &\leq n \cdot (d + 1) \cdot \text{poly}(\lambda) + m \end{aligned} \tag{16}$$

as desired.

6. Output  $(\mathbf{v}, \pi)$ .

$\mathcal{V}(\text{crs}, \mathbf{v}, \pi) \rightarrow 0/1$ . This polynomial time algorithm does the following:

1. Parse  $\text{crs} = (\text{crs}_1, \dots, \text{crs}_\lambda, \text{hk}_1, \dots, \text{hk}_\lambda)$ .
2. Let  $d \in [\lambda]$  be the depth of  $\mathbf{v}$ . If  $d \geq \lambda$  then output 0.
3. Output  $\text{seBARG}.\mathcal{V}(\text{crs}_{d+1}, (Z^{(1)}, \dots, Z^{(k)}), \pi)$ , where  $Z^{(i)} = ((\text{crs}_j)_{j \in [d]}, (\text{hk}_j)_{j \in [d+1]}, \mathbf{v}, i)$ .

$\text{Extract}(\text{td}, \mathbf{v}, I) \rightarrow x$ . This polynomial time algorithm does the following:

1. Parse  $\text{td} = (\text{td}_1, \dots, \text{td}_\lambda, \text{SEH.td}_1, \dots, \text{SEH.td}_\lambda)$  and  $I = (i_1, \dots, i_\lambda)$ .
2. Let  $d \in [\lambda]$  be the depth of  $\mathbf{v}$ . If  $d \geq \lambda$  then output 0.
3. Let  $x_{i_d} = \text{SEH.Extract}(\text{SEH.td}_d, \mathbf{v})$ .
4. For  $j \in [d-1]$ : Compute  $x_{(i_d, \dots, i_{d-j})} = \text{SEH.Extract}(\text{SEH.td}_{d-j}, x_{(i_d, \dots, i_{d-j+1})})$ .
5. Output  $x_{(i_d, \dots, i_1)}$ .

**Theorem 6.6.** *The above construction is a hashed multi-BARG scheme (according to Definition 6.5).*

*Proof.* In what follows we prove that our hashed multi-BARG satisfies all the desired properties:

**Efficiency.** Follows from the construction, together with the fact that the underlying seBARG scheme and SEH family are both rate-1.

**Completeness.** Follows directly from the completeness of the underlying seBARG scheme and the opening completeness of the underlying SEH family, and from the fact that the underlying single-seBARG scheme is rate-1.

**Index-hiding.** Follows immediately from the index-hiding property of the underlying single-seBARG scheme and the SEH family (via a standard hybrid argument).

**Extraction correctness.** Follows immediately from extraction correctness of the underlying SEH family.

**Somewhere argument of knowledge.** Let  $\text{seBARG}.\mathcal{E}$  denote the PPT extractor corresponding to the underlying seBARG scheme. We define an extractor  $\mathcal{E}$  that takes as input  $(\text{td}, \mathbf{v}, \pi)$  and does the following:

1. Parse  $\text{td} = (\text{td}_1, \dots, \text{td}_\lambda, \text{SEH.td}_1, \dots, \text{SEH.td}_\lambda)$ .
2. Let  $d \in [\lambda]$  be the depth of  $\mathbf{v}$ , and denote by  $\mathbf{v}^{(d)} = \mathbf{v}$  and  $\pi^{(d)} = \pi$ .
3. For every  $i \in [d-1]$  compute by backward induction the triplet

$$\left( \mathbf{v}^{(i)}, \rho_i, \pi^{(i)} \right) = \text{seBARG}.\mathcal{E} \left( \text{td}_{i+1}, \left( Z^{(j)} \right)_{j \in [k]}, \pi^{(i+1)} \right)$$

where

$$Z^{(j)} = \left( (\text{crs}_\ell)_{\ell \in [d]}, (\text{hk}_\ell)_{\ell \in [d+1]}, \mathbf{v}^{(i+1)}, j \right)$$

and let  $(x, \rho, w) = \text{seBARG}.\mathcal{E} \left( \text{td}_1, (\text{hk}_1, \mathbf{v}_1, j)_{j \in [k]}, \pi^{(i)} \right)$ .

4. Output  $(x, w)$ .

We need to argue that for any poly-size adversary  $\mathcal{A}$  there exists a negligible function  $\text{negl}$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, \text{hk}, \mathbf{v}, \pi) = 1 \\ \wedge x = \text{SEH.Extract}(\text{SEH.td}, \mathbf{v}, i) \\ \wedge w \text{ is not a valid witness for } x \in \mathcal{L} \end{array} : \begin{array}{l} (\text{crs}, \text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)) \\ (\mathbf{v}, \pi) = \mathcal{A}(\text{crs}, \text{hk}) \\ (x, w) \leftarrow \mathcal{E}(\text{td}, \mathbf{v}, \pi) \\ \text{parse td} = (\text{seBARG.td}, \text{SEH.td}) \end{array} \right] \leq \text{negl}(\lambda). \quad (17)$$

We prove this by induction on the depth  $d$  of  $\mathbf{v}$  output by  $\mathcal{A}$ .

**Induction base:** Fix any poly-size adversary that always outputs  $(\mathbf{v}, \pi)$  such that  $d(\mathbf{v}) = 1$ . In this case, the equation above follows immediately from the somewhere argument of knowledge property of the underlying seBARG scheme together with the somewhere statistically binding w.r.t. opening property of the underlying SEH family.

**Induction step:** Suppose that Equation (17) holds for every poly-size  $\mathcal{A}$  that outputs  $(\mathbf{v}, \pi)$  such that  $d(\mathbf{v}) < d$ , and we prove that it holds for any poly-size  $\mathcal{A}$  that outputs  $(\mathbf{v}, \pi)$  such that  $d(\mathbf{v}) \leq d$ . Recall that  $\mathcal{E}(\text{td}, \mathbf{v}, \pi)$  first computes

$$\left( \mathbf{v}^{(d-1)}, \rho_{d-1}, \pi^{(d-1)} \right) = \text{seBARG.}\mathcal{E} \left( \text{td}_d, \left( Z^{(j)} \right)_{j \in [k]}, \pi \right)$$

where

$$Z^{(j)} = \left( (\text{crs}_\ell)_{\ell \in [d]}, (\text{hk}_\ell)_{\ell \in [d+1]}, \mathbf{v}, j \right)$$

By the somewhere argument of knowledge property of the underlying seBARG scheme there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, \text{hk}, \mathbf{v}, \pi) = 1 \\ \wedge \left( \mathbf{v}^{(d-1)}, \rho_{d-1}, \pi^{(d-1)} \right) \text{ is not} \\ \quad \text{a valid witness for } Z^{(i_d)} \in \mathcal{L}' \end{array} : \begin{array}{l} (\text{crs}, \text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)) \\ \text{parse td} = (\text{seBARG.td}, \text{SEH.td}) \end{array} \right] \leq \mu(\lambda)$$

By the definition of  $\mathcal{L}'$  this implies that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}(\text{crs}, \text{hk}, \mathbf{v}, \pi) = 1 \\ \wedge \left( \text{SEH.Verify}(\text{hk}, \mathbf{v}, I_{i_d}, \mathbf{v}^{(d-1)}, \rho_{d-1}) = 0 \right. \\ \quad \left. \vee \text{seBARG.}\mathcal{V}(\text{crs}_d, Y, \pi^{(d-1)}) = 0 \right) \end{array} : \begin{array}{l} (\text{crs}, \text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, n, k, (i_1, \dots, i_\lambda)) \\ \text{parse td} = (\text{seBARG.td}, \text{SEH.td}) \end{array} \right] \leq \mu(\lambda)$$

where  $Y = (Y^{(1)}, \dots, Y^{(k)})$  and for  $j \in [k]$ ,

$$Y^{(j)} = \left( (\text{crs}_\ell)_{\ell \in [d-1]}, (\text{hk}_\ell)_{\ell \in [d]}, \mathbf{v}^{(d-1)}, j \right).$$

This together with our induction hypothesis implies Equation (17), as desired.  $\square$

## 7 Applications

### 7.1 Aggregate Signatures

Our first application is to show how multi-hop seBARGs can be used to build an unbounded (multi-hop, multi-signer) aggregate signature scheme.

#### 7.1.1 Definition

The notion of (multi-signer) aggregate signatures as introduced by Boneh, Gentry, Lynn, and Shacham [BGLS03] is a digital signature scheme that comes with two poly-time algorithms, **Aggregate** and **AggVerify**, where **Aggregate** is used to aggregate an arbitrary polynomial number of signatures  $\{\sigma_i\}$  corresponding to messages  $\{m_i\}$  generated using verification keys  $\{vk_i\}$ , into a shorter aggregate signature  $\hat{\sigma}$ , and **AggVerify** can be used to verify such aggregate signatures with respect to the sequence of messages and verification keys.

We want aggregated signatures which can be further aggregated; i.e., that aggregation can be performed in multiple hops or sequentially, and all the key-message-signature tuples need not be available at once. Indeed, many existing schemes in the literature give multi-hop aggregation by default, yet this notion was not formalized. In this work, we give a formal definition of a multi-hop aggregate signature scheme.

**Syntax.** A signature scheme  $\mathcal{S}$  for message space  $\{\{0, 1\}^\lambda\}_{\lambda \in \mathbb{N}}$  consists of the following polynomial time algorithms:

**Setup** $(1^\lambda) \rightarrow (vk, sk)$ . The setup algorithm, on input the security parameter  $1^\lambda$ , outputs a pair of signing and verification keys  $(vk, sk)$ .

**Sign** $(sk, m) \rightarrow \sigma$ . The signing algorithm takes as input a signing key  $sk$  and a message  $m \in \{0, 1\}^\lambda$ , and computes a signature  $\sigma$ .

**Verify** $(vk, m, \sigma) \rightarrow 0/1$ . The verification algorithm takes as input a verification key  $vk$ , a message  $m \in \{0, 1\}^\lambda$ , and a signature  $\sigma$ . It outputs a bit to signal whether the signature is valid or not.

**Definition 7.1.** A signature scheme  $\mathcal{S} = (\text{Setup}, \text{Sign}, \text{Verify})$  for message space  $\{\{0, 1\}^\lambda\}_{\lambda \in \mathbb{N}}$  is required to satisfy the following properties:

**Correctness.** For any  $\lambda \in \mathbb{N}$ , any  $(vk, sk) \leftarrow \text{Setup}(1^\lambda)$ , and any  $m \in \{0, 1\}^\lambda$ ,

$$\Pr [ \text{Verify}(vk, m, \sigma) = 1 \quad : \quad \sigma \leftarrow \text{Sign}(sk, m) ] = 1.$$

**Unforgeability.** For every admissible poly-size attacker  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for all  $\lambda \in \mathbb{N}$ ,

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

and  $\mathcal{A}$  is admissible if it did not query  $m^*$  to the  $\text{Sign}(sk, \cdot)$  oracle.

**Syntax.** A (multi-hop, multi-signer) aggregate signature scheme  $\mathcal{S}$  for message space  $\{\{0, 1\}^\lambda\}_{\lambda \in \mathbb{N}}$  consists of a signature scheme (Setup, Sign, Verify) and the following polynomial time algorithms:

$\text{CRS}(1^\lambda) \rightarrow \text{crs}$ . The CRS generation algorithm samples global parameters  $\text{crs}$ .

$\text{Aggregate}(\text{crs}, \{(T_i, \sigma_i)\}_i) \rightarrow \hat{\sigma}/\perp$ . The signature aggregation algorithm takes as input a  $\text{crs}$ , a sequence of (verification-)key-message trees along with a (possibly aggregated) signature  $\sigma_i$ . It outputs either an aggregated signature  $\hat{\sigma}$  or a special abort symbol  $\perp$ .

**Remark 7.1.** *Similar to the case of multi-hop BARGs, we define a (verification-)key-message tree  $T$  of key-message pairs as a tree of varying arity each of whose leaf nodes is associated with a single key-message pair, and each intermediate node corresponds to an aggregated signature that certifies the validity for all key-message pairs associated with its children. As before, the signatures at the intermediate nodes are not stored in the tree. (See definition 6.1.)*

$\text{AggVerify}(\text{crs}, T, \hat{\sigma}) \rightarrow 0/1$ . The aggregated verification algorithm takes as input a  $\text{crs}$ , a key-message tree  $T$ , and an aggregate signature  $\hat{\sigma}$ . It outputs a bit to signal whether the aggregated signature  $\hat{\sigma}$  is valid or not.

**Definition 7.2.** *A multi-hop, multi-signer aggregate signature scheme  $\mathcal{S} = (\text{CRS}, \text{Setup}, \text{Sign}, \text{Verify}, \text{Aggregate}, \text{AggVerify})$  for message space  $\{\{0, 1\}^\lambda\}_{\lambda \in \mathbb{N}}$  is required to satisfy the following properties, in addition to (Setup, Sign, Verify) being a signature scheme for message space  $\{\{0, 1\}^\lambda\}_{\lambda \in \mathbb{N}}$ :*

**Compactness of aggregation.** *The size of an aggregated signature  $\hat{\sigma}$  corresponding to a key-message-tree  $T$  of depth  $d$  is  $|\hat{\sigma}| = \max_i |\sigma_i| + d \cdot \text{poly}(\lambda, \log|T|)$ , where  $|\sigma_i| = \text{poly}(\lambda)$  is the length of a valid signature corresponding to leaf  $(\text{vk}_i, m_i)$  in  $T$ .*

**Aggregated correctness.** *For any  $\lambda \in \mathbb{N}$  and  $\ell, N \leq 2^\lambda$ , any  $\text{crs} \leftarrow \text{CRS}(1^\lambda)$ , any  $(\text{vk}_j, \text{sk}_j) \leftarrow \text{Setup}(1^\lambda)$  for  $j \in [N]$ , any key mapping function  $\pi : [\ell] \rightarrow [N]$ <sup>17</sup>, any messages  $m_1, \dots, m_\ell \in \{0, 1\}^\lambda$ , and any key-message tree  $T$  with  $\ell$  leaves corresponding to key-message pairs  $(\text{vk}_{\pi(i)}, m_i)_{i \in [\ell]}$ ,*

$$\Pr \left[ \text{AggVerify}(\text{crs}, T, \hat{\sigma}) = 1 \quad : \quad \hat{\sigma} \leftarrow \text{Agg} \left( \text{crs}, (\text{vk}_j, \text{sk}_j)_{j \in [N]}, T \right) \right] = 1$$

where  $\text{Agg} \left( \text{crs}, (\text{vk}_j, \text{sk}_j)_{j \in [N]}, T \right)$  is defined by induction on  $d = \text{depth}(T)$  as follows:

If  $d = 1$  then parse  $T = (T_1 = (\text{vk}_{\pi(1)}, m_1), \dots, T_\ell = (\text{vk}_{\pi(\ell)}, m_\ell))$  and output

$$\hat{\sigma} = \text{Aggregate}(\text{crs}, \{(T_i, \sigma_i \leftarrow \text{Sign}(\text{sk}_{\pi(i)}, m_i))\}_{i \in [\ell]}).$$

If  $d > 1$  then parse  $T = (T_1, \dots, T_k)$  and output

$$\hat{\sigma} = \text{Aggregate}(\text{crs}, \{(T_i, \sigma_i \leftarrow \text{Agg}((\text{vk}_j, \text{sk}_j)_{j \in [N]}, T_i))\}_{i \in [k]}).$$

<sup>17</sup>The goal behind the key mapping function is to allow aggregation of multiple signatures that come from a single-signer. That is, an aggregator might want to aggregate signatures from multiple signers, where each signer might be contributing more than one signature potentially.

**Aggregated unforgeability.** For every admissible poly-size attacker  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$  and any poly-size tree  $T$ ,

$$\Pr \left[ \text{AggVerify}(\text{crs}, T^*, \hat{\sigma}^*) = 1 \quad : \quad \begin{array}{l} \text{crs} \leftarrow \text{CRS}(1^\lambda), (\text{vk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (T^*, \hat{\sigma}^*) = \mathcal{A}(\text{crs}, \text{vk}) \end{array} \right] \leq \text{negl}(\lambda).$$

and  $\mathcal{A}$  is admissible if  $T^*$  is consistent with  $T$  (i.e., they have the same structure, see Notation 6.2 for a formal definition) and contains a leaf  $(\text{vk}_i, m_i)$  such that  $\text{vk}_i = \text{vk}$  and  $m_i$  was not queried by  $\mathcal{A}$  to the  $\text{Sign}(\text{sk}, \cdot)$  oracle.

### 7.1.2 Construction and Analysis

In this section we upgrade any signature scheme  $S = (\text{Setup}, \text{Sign}, \text{Verify})$  to a multi-hop, multi-signer aggregate signature scheme, using any multi-hop seBARG scheme

$$\text{multi-BARG} = (\text{multi-BARG.Gen}, \text{multi-BARG.P}, \text{multi-BARG.AggProve}, \text{multi-BARG.V})$$

for the NP language

$$\mathcal{L}_S = \{x = (\text{vk}, m) : \exists \sigma \text{ s.t. } \text{Verify}(\text{vk}, m, \sigma) = 1\}.$$

We define the CRS, Aggregate, and AggVerify algorithms below.

$\text{CRS}(1^\lambda) \rightarrow \text{crs}$ . This algorithm simply runs the  $\text{multi-BARG.Gen}$ . It sets the maximum number of hops to be  $\lambda$  (refer to Remark 7.2 below for details on why  $\lambda$  hops is sufficient for unbounded signature aggregation), and chooses the extraction indices arbitrarily, e.g., as all ones. Namely, it samples

$$(\text{crs}, \text{td}) \leftarrow \text{multi-BARG.Gen}(1^\lambda, d = \lambda, I = (1, \dots, 1)).$$

and outputs  $\text{crs}$ .

$\text{Aggregate}(\text{crs}, \{(T_i, \sigma_i)\}_{i \in [\ell]}) \rightarrow \hat{\sigma}/\perp$ . The signature aggregator first checks that for all  $i \in [\ell]$ ,  $T_i$  is a well-formed key-message tree and  $\sigma_i$  is a well-formed (possibly aggregated) signature. If not, it outputs  $\perp$ , and if so, it simply runs the  $\text{multi-BARG}$  proof combiner. Namely, it outputs the aggregated signature

$$\hat{\sigma} \leftarrow \text{multi-BARG.AggProve}(\text{crs}, T_1, \dots, T_\ell, \sigma_1, \dots, \sigma_\ell).$$

$\text{AggVerify}(\text{crs}, T, \hat{\sigma}) \rightarrow 0/1$ . The aggregated verifier simply runs the  $\text{multi-BARG}$  verifier. Namely, it outputs the bit

$$b = \text{multi-BARG.V}(\text{crs}, T, \hat{\sigma}).$$

**Remark 7.2** (Unbounded Aggregation using Tree-Based Aggregation). Note that in the above description we set the number of hops supported by the  $\text{multi-BARG}$  scheme to be  $\lambda$ . This allows to aggregate  $2^\lambda$  signatures by aggregating via a binary tree  $T$ . Namely, when aggregating signatures we will ensure that the key-message tree  $T$  has the property that if the depth of  $T$  is  $d$  then the number of leaves in  $T$  (i.e., the number of signatures that are aggregated) is at least  $2^{d-1}$ . This is done by ensuring (inductively) that any node at any level  $i \in [d]$  of the tree, satisfies that its left sub-tree is a complete binary tree (with  $2^{i-1}$  leaves).

**Theorem 7.3.** *Assuming hardness of the LWE problem, if  $S = (\text{Setup}, \text{Sign}, \text{Verify})$  is a signature scheme, then  $(\text{CRS}, \text{Setup}, \text{Sign}, \text{Verify}, \text{Aggregate}, \text{AggVerify})$  is a multi-hop, multi-signer aggregate signature scheme as in Definition 7.2.*

*Proof.* In what follows we show our construction satisfies all the desired properties.

**Compactness of aggregation.** Follows from the efficiency of the underlying multi-hop seBARG scheme multi-BARG.

**Correctness.** Follows from the correctness of the underlying signature scheme  $S$ .

**Unforgeability.** Follows from the unforgeability of the underlying signature scheme  $S$ .

**Aggregated correctness.** Follows from the correctness of the underlying signature scheme  $S$  and the completeness of the underlying multi-hop seBARG scheme multi-BARG.

**Aggregated unforgeability.** Suppose there exists an admissible poly-size attacker  $\mathcal{A}$ , a poly-size tree  $T$ , and a non-negligible function  $\epsilon$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{AggVerify}(\text{crs}, T^*, \hat{\sigma}^*) = 1 \\ \text{crs} \leftarrow \text{CRS}(1^\lambda), \\ (\text{vk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (T^*, \hat{\sigma}^*) = \mathcal{A}(\text{crs}, \text{vk}) \end{array} \right] \geq \epsilon(\lambda). \quad (18)$$

Recall that an attacker is admissible if  $T^*$  is consistent with  $T$  and contains a leaf  $(\text{vk}_{i^*}, m_{i^*})$  such that  $\text{vk}_{i^*} = \text{vk}$  and  $m_{i^*}$  was not queried by  $\mathcal{A}$  to the  $\text{Sign}(\text{sk}, \cdot)$  oracle. Let  $\ell = \text{poly}(\lambda)$  be the number of leaves in the tree  $T$  (and thus also in the tree  $T^*$ ). For any  $i \in [\ell]$ , let  $\text{CRS}_i$  be identical to  $\text{CRS}$  except that it sets  $i_1, \dots, i_\lambda$  so that the signature  $\sigma_i$  corresponding to leaf  $(\text{vk}_i, m_i)$  is extractable. In addition,  $\text{CRS}_i$  outputs the multi-BARG trapdoor  $\text{td}$ . By the index hiding property of the multi-BARG scheme, Equation (18) implies that there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{AggVerify}(\text{crs}, T^*, \hat{\sigma}^*) = 1 \\ \wedge i^* = i \\ i \stackrel{\$}{\leftarrow} [\ell], (\text{crs}, \text{td}) \leftarrow \text{CRS}_i(1^\lambda), \\ (\text{vk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (T^*, \hat{\sigma}^*) = \mathcal{A}(\text{crs}, \text{vk}) \end{array} \right] \geq \frac{1}{\ell} \cdot \epsilon(\lambda) - \mu(\lambda). \quad (19)$$

Let  $\mathcal{E}$  be the multi-BARG extractor. The somewhere argument of knowledge property of multi-BARG implies that there exists a negligible function  $\nu$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{AggVerify}(\text{crs}, T^*, \hat{\sigma}^*) = 1 \\ \wedge \text{Verify}(\text{vk}_i, m_i, \sigma) \neq 1 \\ i \stackrel{\$}{\leftarrow} [\ell], (\text{crs}, \text{td}) \leftarrow \text{CRS}_i(1^\lambda), \\ (\text{vk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (T^*, \hat{\sigma}^*) = \mathcal{A}(\text{crs}, \text{vk}), \\ \sigma = \mathcal{E}(\text{td}, T^*, \hat{\sigma}^*) \end{array} \right] \leq \nu(\lambda). \quad (20)$$

Equations (19) and (20) together imply that there exists a non-negligible function  $\epsilon'$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} \text{AggVerify}(\text{crs}, T^*, \hat{\sigma}^*) = 1 \\ \wedge i^* = i \\ \wedge \text{Verify}(\text{vk}, m_{i^*}, \sigma) = 1 \\ i \stackrel{\$}{\leftarrow} [\ell], (\text{crs}, \text{td}) \leftarrow \text{CRS}_i(1^\lambda), \\ (\text{vk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (T^*, \hat{\sigma}^*) = \mathcal{A}(\text{crs}, \text{vk}), \\ \sigma = \mathcal{E}(\text{td}, T^*, \hat{\sigma}^*) \end{array} \right] \geq \epsilon'(\lambda),$$

but this contradicts the unforgeability of the underlying signature scheme  $S$ , since  $m_{i^*}$  was never queried to the  $\text{Sign}(\text{sk}, \cdot)$  oracle.  $\square$

## 7.2 Incrementally Verifiable Computation

In this section, we construct an incrementally verifiable computation (IVC) scheme for deterministic computations with proof size  $\text{poly}(\lambda, \log T)$  using our rate-1 hashed multi-hop BARG scheme (see Section 6) associated with a rate-1 fully-local somewhere extractable hash function. Intuitively, an (IVC) scheme is simply an *updatable* SNARG. Below we recall the formal definition of IVC.

### 7.2.1 Definition

**Language  $\mathcal{L}_{\mathcal{M}}$ .** For any deterministic Turing machine  $\mathcal{M}$  with run-time  $T = T(n)$  and configuration size  $S = S(n)$ , any input  $z \in \{0, 1\}^n$ , and any  $t \in [T(n)]$ , we denote by  $\mathcal{M}(z; 1^t) \in \{0, 1\}^{S(n)}$  the configuration of  $\mathcal{M}$  when executed on input  $z$  after  $t$  steps. We consider the following language:

$$\mathcal{L}_{\mathcal{M}} = \left\{ (z, t, c) : t \in [T(|z|)] \wedge \mathcal{M}(z; 1^t) = c \in \{0, 1\}^{S(|z|)} \right\} \quad (21)$$

**Notation 7.4.** We use  $\text{nxt-cnfg}_{\mathcal{M}}(\cdot)$  as the shorthand denoting the computation of the next configuration from any given valid intermediate configuration of the Turing machine  $\mathcal{M}$  and an input  $z$ . That is,  $c_0 = \text{nxt-cnfg}_{\mathcal{M}}(\emptyset, z)$  is the starting configuration, and  $c_i = \text{nxt-cnfg}_{\mathcal{M}}(c_{i-1}, z)$  for  $i \geq 1$ . It follows by induction that  $c_i = \mathcal{M}(z; 1^i)$  for  $i \geq 1$ . We refer to  $c_0$  as the initial configuration of the machine, and  $z$  represents the fixed input tape of the Turing machine.

**Syntax.** An incrementally verifiable computation (IVC) scheme for a Turing machine  $\mathcal{M}$  consists of the following polynomial time algorithms:

**Gen**( $1^\lambda, n, 1^S$ )  $\rightarrow$  **crs**. The probabilistic setup algorithm takes as input a security parameter  $1^\lambda$ , the input length  $n$ , the maximum configuration size  $S = S(n)$  in unary, and outputs a common reference string **crs**.

**Update**(**crs**,  $z, c_{t-1}, \pi_{t-1}$ )  $\rightarrow$  ( $c_t, \pi_t$ ). The update algorithm takes as input a CRS **crs**, an input  $z \in \{0, 1\}^n$ , an intermediate configuration  $c_{t-1}$ , and a proof  $\pi_{t-1}$ . It outputs the next configuration  $c_t$  and an updated proof  $\pi_t$ .

**V**(**crs**,  $x = (z, t, c_t), \pi$ )  $\rightarrow$  0/1. The verifier algorithm takes as input a CRS **crs**, an instance  $x = (z, t, c_t)$ , and a proof  $\pi$ . It outputs a bit to signal whether the proof is valid or not.

**Definition 7.5.** An incremental verifiable computation scheme (**Gen**, **P**, **Update**, **V**) for  $\mathcal{M}$  is required to satisfy the following properties:

**Efficiency.** The running time of the setup algorithm is at most  $\text{poly}(\lambda, S, \log n, \log T(n))$ , the run-time of verifier and update algorithm is at most  $\text{poly}(\lambda, S, n, \log T(n))$ , and the prover runs in time at most  $t \cdot \text{poly}(\lambda, S, n, \log T(n))$ .

**Completeness.** For every  $\lambda, n \in \mathbb{N}$  any  $z \in \{0, 1\}^n$ , and any time step  $t \in [T(n)]$ ,

$$\Pr \left[ \mathcal{V}(\text{crs}, (z, t, c), \pi) = 1 \quad : \quad \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, n, 1^{S(n)}) \\ (c, \pi) \leftarrow \mathcal{P}(\text{crs}, z, 1^t) \end{array} \right] = 1.$$



where  $\mathcal{P}(\text{crs}, z, 1^t)$  is defined by induction on  $t$  as follows:

If  $t = 1$  then let  $c_0 = \text{nxt-cnfg}_{\mathcal{M}}(\emptyset, z)$  and  $\pi_0 = \emptyset$  and output

$$(c, \pi) = \text{Update}(\text{crs}, z, c_0, \pi_0).$$

If  $t > 1$  then inductively compute  $(c_{t-1}, \pi_{t-1}) = \mathcal{P}(\text{crs}, z, 1^{t-1})$  and output

$$(c, \pi) = \text{Update}(\text{crs}, z, c_{t-1}, \pi_{t-1}).$$

**Soundness.** For any poly-size adversary  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  there exists a negligible function  $\text{negl}(\cdot)$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge x \notin \mathcal{L}_{\mathcal{M}} \\ \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ \text{crs} \leftarrow \text{Gen}(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}) \end{array} \right] \leq \text{negl}(\lambda).$$

## 7.2.2 Construction and Analysis

In this section, we construct an IVC scheme from a specific instantiation of our hashed multi-BARG construction. Let

$$(\text{multi-BARG.Gen}, \text{multi-BARG.P}, \text{multi-BARG.AggProve}, \text{multi-BARG.V})$$

be our hashed multi-BARG construction as in Section 6.2 corresponding to the NP language

$$\mathcal{L}_{\text{nxt-cnfg}, \mathcal{M}} = \{(z, c, c') \mid c' = \text{nxt-cnfg}_{\mathcal{M}}(c, z)\},$$

associated with a “segmented” rate-1 fully-local somewhere extractable hash family

$$(\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify}, \text{SEH.Extract}),$$

defined below, which can be trivially instantiated from any rate-1 flSEH family.

**Segmented hashes.** Given hash key  $\text{hk}_1$  and instances  $x_i = (z, c_{i-1}, c_i)$  for  $i \in [2k]$ ,  $\text{SEH.Hash}$  outputs a segmented hash value  $(\mathbf{v}, \mathbf{v}')$  of depth 1 where  $\mathbf{v} = \text{SEH.Hash}(\text{hk}_1, z, c_0, c_1, \dots, c_k)$  and  $\mathbf{v}' = \text{SEH.Hash}(\text{hk}_1, z, c_k, c_{k+1}, \dots, c_{2k})$ . That is, although each configuration  $c_i$  is included in two instances, it is only included once in the string being hashed (except  $c_k$ ). An opening of  $c_i$  w.r.t.  $(\mathbf{v}, \mathbf{v}')$  consists of an opening w.r.t.  $\mathbf{v}$  if  $0 \leq i < k$ , an opening w.r.t.  $\mathbf{v}'$  if  $k < i \leq 2k$ , and openings w.r.t. both  $\mathbf{v}$  and  $\mathbf{v}'$  if  $i = k$ . We set the hash parameters so  $\mathbf{v}$  and  $\mathbf{v}'$  are each statistically binding on the input  $z$  and two configurations.

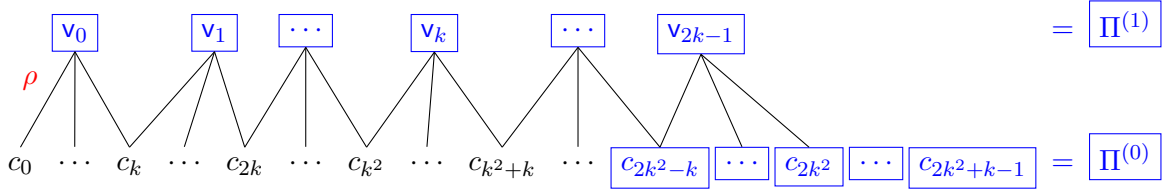
Similarly, given hash key  $\text{hk}_{d+1}$  and hash values  $(\mathbf{v}_{i-1}, \mathbf{v}_i)$  of depth  $d$  for  $i \in [2k]$ ,  $\text{SEH.Hash}$  outputs a segmented hash value  $(\mathbf{v}, \mathbf{v}')$  of depth  $d + 1$  where  $\mathbf{v} = \text{SEH.Hash}(\text{hk}_{d+1}, \mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_k)$  and  $\mathbf{v}' = \text{SEH.Hash}(\text{hk}_{d+1}, \mathbf{v}_k, \mathbf{v}_{k+1}, \dots, \mathbf{v}_{2k})$ . We set the hash parameters so  $\mathbf{v}$  and  $\mathbf{v}'$  are each statistically binding on one hash value of depth  $d$ .

**Overview.** We aggregate in a tree-like structure, and only aggregate configurations (leaves of the tree) or hash values (non-leaf nodes of the tree) in batches of  $2k$ . We store all the unaggregated configurations/hash values at each level of the tree while waiting to accumulate enough configurations/hash values to aggregate. In our construction, a proof  $\pi = (\rho, \Pi^{(0)}, \Pi^{(1)}, \dots, \Pi^{(\lambda)})$  contains an opening  $\rho$ , a list  $\Pi^{(0)}$  of configurations, and, for every  $d \in \lambda$ , a list  $\Pi^{(d)}$  of segmented hash values of depth  $d$  (and corresponding proofs). Each proof certifies that one segmented hash value hashes a sequence of  $2k^d$  consecutive configurations.

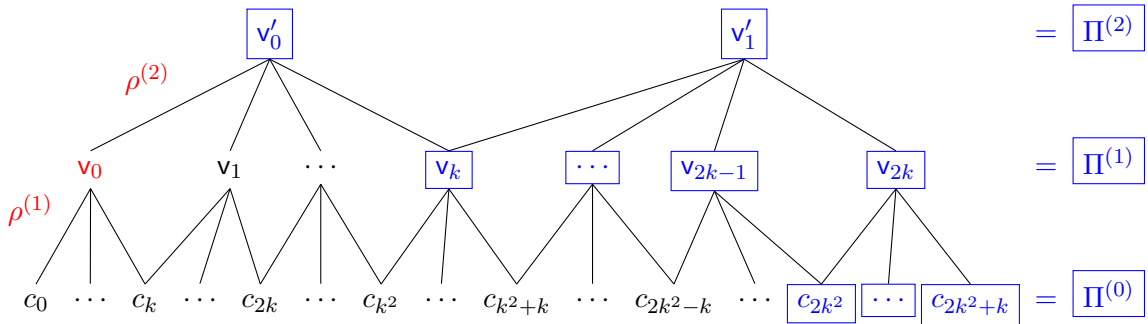
Since we need to ensure that the last configuration in one batch and the first configuration in the next batch are consecutive, we need to overlap our batches. The list  $\Pi^{(d)}$  has the property that every segmented hash value has its first component in common with the immediately previous segmented hash value in  $\Pi^{(d)}$  and its second component in common with the immediately subsequent segmented hash value in  $\Pi^{(d)}$ . In our construction, we only need to store one copy of each of these components.

Initially,  $\rho$  and the lists are empty. As we keep updating  $\pi$  for more timesteps, whenever a list reaches length  $2k$ , we aggregate, add the aggregated hash value/proof to the list one level higher, and then erase the first half of the list. The second half of the list will be aggregated again once the list grows (so we can ensure that we're always aggregating consecutive batches of configurations). We also maintain  $\rho$  so that it always opens the “leftmost” hash value, i.e., the first hash value stored in the highest nonempty level, to the initial configuration  $c_0$ .

For example, after updating the proof  $\pi$  for timestep  $2k^2 + k - 1$ , the hash tree looks like this (only the blue boxes are stored in  $\pi$ ):



Note that  $\Pi^{(1)}$  also contains proofs  $\{\pi_i\}_{i=1}^{2k-1}$  for instances  $\{(v_{i-1}, v_i)\}_{i=1}^{2k-1}$ , which are omitted from the above figure for simplicity. After updating the proof  $\pi$  for the next timestep  $2k^2 + k$ , the hash tree looks like this (again, only the blue boxes are stored in  $\pi$ ):



Note that  $\Pi^{(1)}$  also contains proofs  $\{\pi_i\}_{i=k+1}^{2k}$  for instances  $\{(v_{i-1}, v_i)\}_{i=k+1}^{2k}$ , and  $\Pi^{(2)}$  also contains a proof  $\pi'$  for instance  $(v'_0, v'_1)$ , which are omitted from the above figure for simplicity. The opening  $\rho$ , which must open the “leftmost” hash value  $v'_0$  to the initial configuration  $c_0$ , now consists of  $(\rho^{(1)}, v_0, \rho^{(2)})$ .

We are now ready to define our IVC scheme.

$\text{Gen}(1^\lambda, n, 1^S) \rightarrow \text{crs}$ . This poly-time algorithm does the following:

1. Set  $i_1, \dots, i_\lambda \in [2k]$  arbitrarily. For example, set  $i_1 = \dots = i_\lambda = 1$ .
2. Sample  $(\text{multi-BARG.crs}, \text{hk}, \text{td}) \leftarrow \text{multi-BARG.Gen}(1^\lambda, n + 2S, 2k, (i_1, \dots, i_\lambda))$ .
3. Output  $\text{crs} = (\text{multi-BARG.crs}, \text{hk})$ .

$\text{Update}(\text{crs}, z, c_{t-1}, \pi_{t-1}) \rightarrow (c_t, \pi_t)$ . This poly-time algorithm does the following:

1. Parse  $\text{crs} = (\text{multi-BARG.crs}, \text{hk} = (\text{hk}_1, \dots, \text{hk}_\lambda))$ .
2. Compute the next configuration  $c_t = \text{nxt-cnfg}_{\mathcal{M}}(c_{t-1}, z)$ .
3. Parse  $\pi_{t-1} = (\rho, \Pi^{(0)}, \dots, \Pi^{(\lambda)})$ .
4. Append  $c_t$  to  $\Pi^{(0)}$ .
5. If  $\Pi^{(0)}$  is not full, i.e., contains  $< 2k + 1$  configurations, go to Item 11.
6. Parse the list  $\Pi^{(0)} = (c_{t-2k}, \dots, c_{t-1}, c_t)$ .
7. Compute  $(\mathbf{v}_*, \mathbf{v}'_*, \pi_*) = \text{multi-BARG.P}( \text{crs}, (z, c_{t-2k+i-1}, c_{t-2k+i})_{i \in [2k]} )$ .
8. Compute  $\rho^{(1)} = \text{SEH.Open}(\text{hk}_1, (z, c_{t-2k}, \dots, c_{t-k}), [n + S])$ . Note that the set  $[n + S]$  corresponds to the coordinates of  $(z, c_{t-2k})$ .
9. Erase the first half of  $\Pi^{(0)}$ , i.e., let  $\Pi^{(0)} = (c_{t-k}, \dots, c_{t-1}, c_t)$ .
10. For  $d = 1, \dots, \lambda$ :
  - If  $\Pi^{(d)}$  is empty, i.e.,  $\mathbf{v}_*$  is now the leftmost hash value, then let  $\Pi^{(d)} = (\mathbf{v}_*)$  and append  $(\rho^{(d)}, \mathbf{v}_*)$  to  $\rho$ .
  - Append  $\mathbf{v}'_*$  and  $\pi_*$  to  $\Pi^{(d)}$ .
  - If  $\Pi^{(d)}$  is not full, i.e., contains  $< 2k + 1$  hash values of depth  $d$ , go to Item 11.
  - Parse the list  $\Pi^{(d)} = (\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{2k}, \pi_1, \dots, \pi_{2k})$ . Recall that for  $i \in [2k]$ ,  $\pi_i$  is a proof for instance  $(\mathbf{v}_{i-1}, \mathbf{v}_i)$ .
  - Compute  $((\mathbf{v}_*, \mathbf{v}'_*, \pi_*)) = \text{multi-BARG.AggProve}(\text{crs}, (\mathbf{v}_{i-1}, \mathbf{v}_i)_{i \in [2k]}, (\pi_i)_{i \in [2k]})$ .
  - Compute  $\rho^{(d+1)} = \text{SEH.Open}(\text{hk}_{d+1}, (\mathbf{v}_0, \dots, \mathbf{v}_k), [|\mathbf{v}_0|])$ .
  - Erase the first half of  $\Pi^{(d)}$ , i.e., let  $\Pi^{(d)} = (\mathbf{v}_k, \mathbf{v}_{k+1}, \dots, \mathbf{v}_{2k}, \pi_{k+1}, \dots, \pi_{2k})$ .
11. Output  $(c_t, \pi_t = (\rho, \Pi^{(0)}, \Pi^{(1)}, \dots, \Pi^{(\lambda)}))$ .

$\mathcal{V}(\text{crs}, x = (z, t, c), \pi) \rightarrow 0/1$ . This poly-time algorithm does the following:

1. Parse  $\text{crs} = (\text{multi-BARG.crs}, \text{hk})$ .
2. Parse  $\pi = (\rho, \Pi^{(0)}, \dots, \Pi^{(\lambda)})$ .
3. Parse  $\Pi^{(0)} = (c_{t-\ell}, \dots, c_{t-1}, c_t)$  for some  $\ell \geq 1$ .
4. If  $c_t \neq c$ , output 0.
5. For  $i \in [\ell]$ , if  $c_{t-i+1} \neq \text{nxt-cnfg}_{\mathcal{M}}(c_{t-i}, z)$ , output 0.
6. Let  $c_0 = \text{nxt-cnfg}_{\mathcal{M}}(\emptyset, z)$ .
7. For  $d = 1, \dots, \lambda$ :
  - Parse  $\Pi^{(d)} = (\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_\ell, \pi_1, \dots, \pi_\ell)$  for some  $\ell \geq 1$ .

- For  $i \in [\ell]$ , if  $\text{multi-BARG.V}(\text{crs}, (v_{i-1}, v_i), \pi_i) \neq 1$ , output 0.
  - If  $\Pi^{(d+1)} = \emptyset$ , i.e.,  $v_0$  is the leftmost hash value, go to Item 8.
8. If  $\text{SEH.Verify}(\text{hk}, v_0, [n + S], \rho, (z, c_0)) = 0$ <sup>18</sup>, output 0.
  9. Output 1.

**Theorem 7.6.** *Assuming hardness of the LWE problem,  $(\text{Gen}, \mathcal{P}, \text{Update}, \mathcal{V})$  is an IVC scheme as in Definition 7.5.*

*Proof.* In what follows we show our construction satisfies all the desired properties.

**Efficiency.** Follows from efficiency of the underlying SEH and multi-BARG schemes.

**Completeness.** Follows from the completeness of the underlying multi-BARG scheme.

**Soundness.** Suppose towards contradiction that there exists a poly-size adversary  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  and a non-negligible function  $\epsilon$  such that for every  $\lambda \in \mathbb{N}$ ,

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge x \notin \mathcal{L}_{\mathcal{M}} \\ \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ \text{crs} \leftarrow \text{Gen}(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}) \end{array} \right] \geq \epsilon(\lambda). \quad (22)$$

Let  $c_0^z = \text{nxt-cnfg}_{\mathcal{M}}(\emptyset, z)$  and  $c_i^z = \text{nxt-cnfg}_{\mathcal{M}}(c_{i-1}^z, z)$  for  $i \in [t]$  be  $\mathcal{M}$ 's true configurations when run for  $t$  timesteps on input  $z$ . For  $I = (i_1, \dots, i_\lambda \in [2k])$ , let  $\text{Gen}_I$  be identical to  $\text{Gen}$  except that it sets  $i_1, \dots, i_\lambda$  as above, as opposed to setting  $i_1 = \dots = i_\lambda = 1$ . In addition,  $\text{Gen}_I$  outputs the multi-BARG trapdoor  $\text{td} = (\text{seBARG.td}, \text{SEH.td})$ . By the index hiding property of the multi-BARG scheme, Equation (22) implies that for any  $I$ ,

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge c_t \neq c_t^z \\ \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}_I(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}) \end{array} \right] \geq \epsilon(\lambda). \quad (23)$$

We define an extractor  $\text{Extract}$  which takes as input the multi-BARG trapdoor  $\text{td} = (\text{seBARG.td}, \text{SEH.td})$ , an instance  $x = (z, t, c_t)$ , a proof  $\pi = (\Pi^{(0)}, \Pi^{(1)}, \dots, \Pi^{(\lambda)})$ , and a timestep  $i \in [t]$ . If configurations  $c_{i-1}$  and  $c_i$  are stored in  $\Pi^{(0)}$ ,  $\text{Extract}$  outputs  $(z, c_{i-1}, c_i)$ . If not,  $\text{Extract}$  locates the hash value  $v_i$  which covers timestep  $i$  (i.e.,  $v_i$  is a depth  $d$  hash of configurations including  $c_{i-1}$  and  $c_i$ ), and outputs  $\text{SEH.Extract}(\text{SEH.td}, v_i)$ .

To contradict Equation (23), it suffices to show by induction that for all  $i \in \{0, \dots, t\}$ , if we set  $I = (i_1, \dots, i_\lambda \in [2k])$  so that the hash value  $v_i$  which covers timestep  $i$  is statistically binding on configurations  $(c_{i-1}, c_i)$ , then

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \\ \wedge c_i \neq c_i^z \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}_I(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}), \\ (z', c_{i-1}, c_i) = \text{Extract}(\text{td}, x, \pi, i) \end{array} \right] \leq \text{negl}(\lambda).$$

<sup>18</sup>We are abusing notation here –  $\rho$  is actually a sequence of openings and hash values which can be verified inductively using the sequence of hash keys.

**Induction base.** Since  $\mathcal{V}$  verifies that  $\rho$  opens the leftmost hash value to the true initial configuration, the somewhere statistically binding property of SEH implies that

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \\ \wedge (z' \neq z \vee c_0 \neq c_0^z) \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}_I(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}), \\ (z', c_0, c_1) = \text{Extract}(\text{td}, x, \pi, 1) \end{array} \right] \leq \text{negl}(\lambda),$$

i.e., the probability that  $c_0$  is incorrect is  $\text{negl}(\lambda)$ .

**Induction step.** Since the event that

$$z \in \{0, 1\}^n \wedge (z' \neq z \vee c_{i-1} \neq c_{i-1}^z) \wedge \mathcal{V}(\text{crs}, x, \pi) = 1$$

is detectable by an efficient distinguisher, the inductive hypothesis and the index hiding property of multi-BARG imply that

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \\ \wedge (z' \neq z \vee c_{i-1} \neq c_{i-1}^z) \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}_I(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}), \\ (z', c_{i-1}, c_i) = \text{Extract}(\text{td}, x, \pi, i) \end{array} \right] \leq \text{negl}(\lambda).$$

Let  $\text{multi-BARG.}\mathcal{E}$  be the hashed multi-hop BARG extractor given by the somewhere argument of knowledge property. We define an extractor  $\text{Extract}'$  which behaves identically to  $\text{Extract}$ , except that if configurations  $c_{i-1}$  and  $c_i$  are not stored in  $\Pi^{(0)}$ ,  $\text{Extract}'$  locates the hash value  $v_i$  which covers timestep  $i$  (i.e.,  $v_i$  is a depth  $d$  hash of configurations including  $c_{i-1}$  and  $c_i$ ), immediately previous hash value  $v_{i-1}$ , and corresponding proof  $\pi_i$  and outputs  $\text{multi-BARG.}\mathcal{E}(\text{seBARG.td}, (v_{i-1}, v_i), \pi_i)$ . The somewhere argument of knowledge property of multi-BARG implies that

$$\Pr \left[ \begin{array}{l} z \in \{0, 1\}^n \wedge \mathcal{V}(\text{crs}, x, \pi) = 1 \\ \wedge \left( (z', c_{i-1}, c_i) \neq \text{Extract}(\text{td}, x, \pi, i) \right. \\ \left. \vee c_i \neq \text{next-cnfg}_{\mathcal{M}}(c_{i-1}, z) \right) \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}_1(1^\lambda), \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}_I(1^\lambda, n, 1^{S(n)}), \\ (x = (z, 1^t, c_t), \pi) \leftarrow \mathcal{A}_2(\text{crs}), \\ (z', c_{i-1}, c_i) = \text{Extract}'(\text{td}, x, \pi, i) \end{array} \right] \leq \text{negl}(\lambda),$$

Note that  $c_i$  is only incorrect if  $c_{i-1}$  is incorrect or  $c_i$  is not the next configuration after  $c_{i-1}$ . Thus the probability that  $c_i$  is incorrect is at most  $\text{negl}(\lambda) + \text{negl}(\lambda) = \text{negl}(\lambda)$ .  $\square$

## References

- [BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012. 1
- [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, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120. ACM, 2013. 1, 5, 8

- [BCI<sup>+</sup>13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333. Springer, 2013. [1](#)
- [BCL<sup>+</sup>21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, pages 681–710, 2021. [5](#)
- [BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, pages 1–18, 2020. [5](#)
- [BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO 2014*, pages 276–294, 2014. [5](#)
- [BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 407–437. Springer, Heidelberg, December 2019. [4](#), [11](#), [12](#), [18](#), [21](#), [64](#), [65](#), [66](#)
- [BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. In *Proceedings of Eurocrypt '03*, volume 2656 of *LNCS*, pages 416–432, 2003. [7](#), [8](#), [50](#)
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012. [11](#)
- [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, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 474–482. ACM, 2017. [1](#)
- [BKK<sup>+</sup>18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 709–721. ACM, 2018. [1](#)
- [BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352, 2020. <https://ia.cr/2020/352>. [5](#)

- [BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1480–1498. IEEE Computer Society, 2015. 5
- [But21] Vitalik Buterin. An approximate introduction to how zk-snarks are possible, 2021. 1
- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011. 11
- [CCDW20] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P. Ward. Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522, 2020. <https://ia.cr/2020/1522>. 5
- [CCH<sup>+</sup>19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. 1
- [CJJ21a] Arka Rai Choudhuri, Abhisek Jain, and Zhengzhong Jin. Snargs for  $\mathcal{P}$  from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 68–79. IEEE, 2021. 1, 2, 3, 5, 16, 17
- [CJJ21b] Arka Rai Choudhuri, Abhisek Jain, and Zhengzhong Jin. Non-interactive batch arguments for np from standard assumptions. Cryptology ePrint Archive, Paper 2021/807, 2021. <https://eprint.iacr.org/2021/807>. 1, 3, 5
- [COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, pages 769–793, 2020. 5
- [CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331, 2010. 5
- [CTV13] Stephen Chong, Eran Tromer, and Jeffrey A. Vaughan. Enforcing language semantics using proof-carrying data. *IACR Cryptol. ePrint Arch.*, page 513, 2013. 5
- [CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 371–403, 2015. 5
- [DGI<sup>+</sup>19a] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th*

*Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2019. [4](#)

- [DGI<sup>+</sup>19b] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019. [11](#)
- [FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. [1](#)
- [GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. [1](#)
- [GH19a] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 438–464. Springer, 2019. [4](#)
- [GH19b] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 438–464. Springer, Heidelberg, December 2019. [11](#)
- [Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010. [1](#)
- [GV22] Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation, 2022. [8](#)
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. [1](#)
- [HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 3–34. Springer, 2015. [2](#), [7](#)
- [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. [2](#), [3](#), [4](#), [11](#), [13](#)
- [JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In



- Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 708–721. ACM, 2021. [1](#)
- [KB20] Assimakis Kattis and Joseph Bonneau. Proof of necessary work: Succinct state verification with fairness guarantees. Cryptology ePrint Archive, Report 2020/190, 2020. <https://ia.cr/2020/190>. [5](#)
- [KLVW22] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. Unpublished manuscript, 2022. [1](#), [2](#), [9](#), [10](#), [12](#), [15](#), [17](#), [18](#)
- [KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 91–118, 2016. [9](#), [10](#)
- [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. [1](#), [5](#), [9](#), [17](#)
- [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. [1](#), [5](#), [8](#)
- [KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 565–574, 2013. [1](#)
- [KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014. [1](#)
- [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, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 330–368. Springer, 2021. [1](#)
- [Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012. [1](#)
- [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. [4](#)
- [Mic93] Silvio Micali. Fair public-key cryptosystems. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 113–138. Springer, Heidelberg, August 1993. [5](#)

- [Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 436–453. IEEE Computer Society, 1994. [1](#), [8](#)
- [NT16] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 255–271, 2016. [5](#)
- [OPWW15a] 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*, volume 9452 of *Lecture Notes in Computer Science*, pages 121–145. Springer, 2015. [2](#), [4](#)
- [OPWW15b] 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. [13](#)
- [RR20] Guy N. Rothblum and Ron D. Rothblum. Batch verification and proofs of proximity with polylog overhead. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 108–138. Springer, 2020. [1](#)
- [RRR18] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Efficient batch verification for UP. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPICs*, pages 22:1–22:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. [1](#)
- [Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, pages 1–18, 2008. [5](#), [8](#)
- [WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. *IACR Cryptol. ePrint Arch.*, page 336, 2022. [1](#), [3](#), [5](#)

## A Homomorphic Encryption with Local Compression Compiler

In this section, we build a homomorphic encryption scheme (Gen, Enc, Eval, Dec) along with compression algorithms

$$(\text{CompGen}, \text{Compress}_1, \text{LinEval}, \text{Compress}_2, \text{CompDec})$$

as defined in section [3.4](#). Our construction is based on the rate-1 homomorphic encryption by Brakerski et al. [\[BDGM19\]](#).

**Construction.** Our encryption scheme has a linear decrypt-and-multiply property (see [BDGM19, Definition 2.9]) for message space  $\{0, 1\}$  and circuit class  $\mathcal{C}$ , and the linearly homomorphic packed Regev encryption scheme as in [BDGM19, § 3.3]. Since our usage of the linearly homomorphic packed Regev encryption scheme is more specific than in [BDGM19], thus we use it directly below rather than abstracting out the required properties as in their rate-1 homomorphic encryption construction. Below we describe our homomorphic encryption with the compression compiler.

$\text{Gen}(1^\lambda) \rightarrow (\text{pk}, \text{sk})$ . The setup algorithm runs the FHE setup algorithm to sample the key pair as  $(\text{pk}, \text{sk}) \leftarrow \text{FHE.Setup}(1^\lambda, 1^d)$ .

$\text{Enc}(\text{pk}, \mu) \rightarrow \text{ct}$ . The encryption algorithm is the FHE encryption algorithm and computes the ciphertext as  $\text{ct} \leftarrow \text{FHE.Enc}(\text{pk}, \mu)$ .

$\text{Eval}(\text{pk}, C, \text{ct}) \rightarrow \text{ct}'$ . The evaluation algorithm simply runs the FHE evaluation algorithm as  $\text{ct}' \leftarrow \text{FHE.Eval}(\text{pk}, C, \text{ct})$ .

$\text{Dec}(\text{sk}, \text{ct}) \rightarrow \mu$ . The decryption algorithm is also the FHE decryption algorithm and computes the output as  $\mu = \text{FHE.Dec}(\text{sk}, \text{ct})$ .

$\text{CompGen}(\text{pk}, \text{sk}, 1^\ell) \rightarrow (\text{pk}_c, \text{sk}_c)$ . Let  $q$  be the LWE modulus chosen for the FHE scheme. Also, let  $\chi$  be a  $B$ -bounded error distribution defined on  $\mathbb{Z}$ , and  $n, m$  be the LWE parameters. We discuss later how the parameters are selected. And, let  $\ell_{\text{sk}} = |\text{sk}|$  denote the length of FHE secret key interpreted as a vector over  $\mathbb{Z}_q$ . That is,  $\text{sk} \in \mathbb{Z}_q^{\ell_{\text{sk}}}$ .

The compression parameter generator samples a random matrix  $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ , a secret matrix  $\mathbf{S} \leftarrow \mathbb{Z}_q^{\ell \times n}$ , and an error matrix  $\mathbf{E} \leftarrow \chi^{\ell \times m}$ . It computes matrix  $\mathbf{B}$  as  $\mathbf{B} = \mathbf{SA} + \mathbf{E}$ , and samples the following sequence of  $\ell \times \ell_{\text{sk}}$  Regev ciphertexts:

$$\forall i \in [\ell], j \in [\ell_{\text{sk}}], \quad \text{ck}_{i,j} = (\mathbf{AR}_{i,j}, \mathbf{BR}_{i,j} + \text{sk}_j \cdot \mathbf{g}^\top \otimes \mathbf{e}_i)$$

where  $\mathbf{R}_{i,j} \leftarrow \{0, 1\}^{m \times \lceil \log q \rceil}$ ,  $\text{sk}_j \in \mathbb{Z}_q$  is the  $j$ -th element of the secret key  $\text{sk}$ ,  $\mathbf{g}^\top = (1, 2, \dots, 2^{\lceil \log q \rceil - 1})$ , and  $\mathbf{e}_i \in \mathbb{Z}_q^\ell$  is the  $i$ -th unit vector. It outputs compression key pair  $(\text{pk}_c, \text{sk}_c)$  as

$$\text{pk}_c = (\text{pk}, \{\text{ck}_{i,j}\}_{i,j}), \quad \text{sk}_c = \mathbf{S}.$$

$\text{Compress}_1(\text{pk}_c, \text{ct}, \ell, i \in [\ell]) \rightarrow \text{ct}_{c_1}$ . It parses the compression public key  $\text{pk}_c$  as above, and let  $\text{vectorize} : [\ell] \times \{0, 1\} \rightarrow \{0, 1\}^\ell$  denote the function that takes as input an index  $i \in [\ell]$ , and a bit  $\mu \in \{0, 1\}$ , and outputs a length- $\ell$  vector which contains  $\mu$  in the  $i$ -th position and zeros everywhere else. Thus,  $\text{vectorize}(i, \mu) = \mu \cdot \mathbf{e}_i$  (i.e., the  $i$ -th unit vector multiplied with bit  $\mu$ ).

The initial compression algorithm homomorphically evaluates  $\text{vectorize}(i, \cdot)$  on the ciphertext  $\text{ct}$  to compute  $\text{ct}_{\text{vec}} = \text{FHE.Eval}(\text{pk}, \text{vectorize}(i, \cdot), \text{ct})$ , a sequence of  $\ell$  evaluated ciphertexts  $\text{ct}_{\text{vec}} = (\text{ct}_{\text{vec},1}, \dots, \text{ct}_{\text{vec},\ell})$ . Next, let  $\text{Dec\&Mult}$  be the linear function that takes as input an FHE secret key  $\text{sk} \in \mathbb{Z}_q^{\ell_{\text{sk}}}$ , an FHE ciphertext  $\text{ct}$  encrypting a single bit  $\mu$ , and a scaling factor  $\omega$ , and outputs a scaled approximation  $\mu\omega$  in the clear. We use the fact that  $\text{Dec\&Mult}$  is a linear function in  $\text{sk}$  over  $\mathbb{Z}_q$ . For more details, we refer the reader to [BDGM19, Definition 2.9]. The algorithm now defines  $\ell$  linear function  $f_i = (f_{i,1}, \dots, f_{i,\ell_{\text{sk}}}) \in \mathbb{Z}_q^{\ell_{\text{sk}}}$  for  $i \in [\ell]$  such that

$$\sum_{j \in [\ell_{\text{sk}}]} f_{i,j} \cdot \text{sk}_j = \text{Dec\&Mult}(\text{sk}, \text{ct}_{\text{vec},i}, 2^{\lceil \log q \rceil - 1}).$$

It then homomorphically evaluates the linear functions on the Regev ciphertexts  $\{\mathbf{ck}_{i,j}\}_{i,j}$  as

$$\mathbf{c}_0 = \sum_{i,j} \mathbf{ck}_{i,j,1} \cdot \mathbf{g}^{-1}(f_{i,j}), \quad \mathbf{c}_1 = \sum_{i,j} \mathbf{ck}_{i,j,2} \cdot \mathbf{g}^{-1}(f_{i,j})$$

where  $\mathbf{ck}_{i,j} = (\mathbf{ck}_{i,j,1}, \mathbf{ck}_{i,j,2}) \in \mathbb{Z}_q^{n \times \lceil \log q \rceil} \times \mathbb{Z}_q^{\ell \times \lceil \log q \rceil}$ , and  $\mathbf{g}^{-1}(\cdot)$  is the standard binary expansion function. Note that we get that  $\mathbf{c}_0 \in \mathbb{Z}_q^n$ , and  $\mathbf{c}_1 \in \mathbb{Z}_q^\ell$ . The algorithm outputs the compressed ciphertext as  $\mathbf{ct}_{\mathbf{c}_1} = (\mathbf{c}_0, \mathbf{c}_1)$ . And, it is interpreted as the following sequence of  $(\ell + 1)$  sub-ciphertexts

$$\mathbf{ct}_{\mathbf{c}_1} = (\text{sub-ct}_0, \text{sub-ct}_1, \dots, \text{sub-ct}_\ell), \quad \text{sub-ct}_0 = \mathbf{c}_0 \in \mathbb{Z}_q^n, \quad \text{sub-ct}_i = \mathbf{c}_{1,i} \in \mathbb{Z}_q,$$

where  $\mathbf{c}_{1,i}$  is the  $i$ -th element of the second ciphertext vector.

$\text{LinEval}(\mathbf{pk}_{\mathbf{c}}, \mathbf{ct}_{\mathbf{c}_1}^{(1)}, \dots, \mathbf{ct}_{\mathbf{c}_1}^{(\ell)}) \rightarrow \mathbf{ct}'_{\mathbf{c}_1}$ . It parses the compressed ciphertexts as  $(\ell + 1)$  sub-ciphertexts each (as describe above). That is,  $\mathbf{ct}'_{\mathbf{c}_1} = (\text{sub-ct}_0^{(i)}, \text{sub-ct}_1^{(i)}, \dots, \text{sub-ct}_\ell^{(i)})$  for  $i \in [\ell]$ , and outputs the linearly evaluated ciphertext as

$$\mathbf{ct}'_{\mathbf{c}_1} = \left( \sum_i \text{sub-ct}_0^{(i)}, \sum_i \text{sub-ct}_1^{(i)}, \dots, \sum_i \text{sub-ct}_\ell^{(i)} \right).$$

That is, each resulting sub-ciphertext is computed by simply adding all the individual underlying sub-ciphertexts. Therefore, the linear evaluation algorithm for compressed ciphertexts satisfies both the ‘locality’ and ‘low depth’ properties as defined in section 3.4. Basically, for ensuring the low depth property, it is enough to perform the  $\ell$  sub-ciphertext additions in a tree-like manner.

$\text{Compress}_2(\mathbf{ct}_{\mathbf{c}_1}) \rightarrow \mathbf{ct}_{\mathbf{c}}$ . The final compression algorithm parses the ciphertext into  $\ell + 1$  components as above. That is,  $\mathbf{ct}_{\mathbf{c}_1} = (\text{sub-ct}_0, \text{sub-ct}_1, \dots, \text{sub-ct}_\ell)$ . And, it compresses each component individually as follows:

$$\text{sub-ct}_{\mathbf{c},j} = \text{Compress}_2(\text{sub-ct}_j) = \begin{cases} \text{sub-ct}_j & \text{if } j = 0 \\ \lfloor \text{sub-ct}_j \rfloor_2 & \text{otherwise.} \end{cases}$$

Here  $\lfloor \cdot \rfloor_2 : \mathbb{Z}_q \rightarrow \{0, 1\}$  outputs the most significant bit of the number. That is,  $\lfloor x \rfloor_2 = 1$  for  $x \geq 2^{\lceil \log q \rceil - 1}$  and 0 otherwise. It outputs the fully compressed ciphertext as  $\mathbf{ct}_{\mathbf{c}} = (\text{sub-ct}_{\mathbf{c},0}, \text{sub-ct}_{\mathbf{c},1}, \dots, \text{sub-ct}_{\mathbf{c},\ell})$ .

$\text{CompDec}(\mathbf{sk}_{\mathbf{c}}, \mathbf{ct}_{\mathbf{c}}) \rightarrow (\mu_1, \dots, \mu_\ell)$ . The decryption algorithm for fully compressed ciphertexts parses the compressed secret key  $\mathbf{sk}_{\mathbf{c}} = \mathbf{S}$ , and let  $\mathbf{s}_i^\top$  be the  $i$ -th row of the matrix  $\mathbf{S}$ . It parses the ciphertext as  $\mathbf{ct}_{\mathbf{c}} = (\text{sub-ct}_{\mathbf{c},0}, \text{sub-ct}_{\mathbf{c},1}, \dots, \text{sub-ct}_{\mathbf{c},\ell})$ .

It outputs the message bits  $\mu_i$  as  $\mu_i = (\text{sub-ct}_{\mathbf{c},i} - \lfloor \mathbf{s}_i^\top \cdot \text{sub-ct}_{\mathbf{c},0} \rfloor_2) \bmod 2$  for  $i \in [\ell]$ .

**Security and setting the LWE parameters.** The LWE parameters are set exactly as in [BDGM19]. Also, the proof of correctness, compactness, and security are based on [BDGM19], except we do not have the randomized shift before rounding, thus at the time of proving correctness this leaves a negligible error over the choice of random coins of setup and encryption.