Mergeable SNARGs for Trapdoor Languages and Their Applications

Zvika Brakerski[1], Maya Farber Brodsky[2], Omer Paneth[2], and Tomer Solomon[2]

[1]Weizmann Institute of Science
[2]Tel Aviv University

Abstract

We present a new method for merging short computationally sound proofs (SNARGs). Intuitively, this means going from short proofs of two statements to a short proof of a statement that is a "logical consequence" of the two.

Our work is the first to handle an unbounded polynomial number of recursive merges with arbitrary topology. Unlike prior works, the running time of our security reduction grows only with the depth of the merging process, and not with the associated "tree size", which could be exponentially larger.

Our method is only applicable to so-called "trapdoor languages", where the validity of a statement can be decided in polynomial time given a trapdoor. Importantly, this trapdoor needs not be known for generating or verifying proofs, and is only used in the security reduction. We present constructions from sub-exponential iO and from LWE, where the former yields fully compact proofs, while in the latter the proof size scales with the depth of the merging process.

We show the usefulness of our method by presenting the first CCA1-secure multi-hop fully homomorphic encryption, from either iO or LWE. We also present the first adaptive multi-hop aggregate signature scheme from LWE.

Contents

1 Introduction **1**
- 1.1 Our Results . . . 3
- 1.2 Technical Overview . . . 6
  - 1.2.1 Mergeable arguments from LWE. . . . 7
  - 1.2.2 Mergeable arguments from iO. . . . 10
  - 1.2.3 CCA1 FHE from mergeable arguments. . . . 11
  - 1.2.4 Strong multi-hop aggregate signatures. . . . 11
  - 1.2.5 IVC with universal soundness for trapdoor languages. . . . 12

2 Preliminaries **13**
- 2.1 Functional Batch Arguments . . . 13
- 2.2 Homomorphic Encryption . . . 14
- 2.3 Indistinguishability Obfuscation . . . 16
- 2.4 Pseudorandom Generator . . . 16
- 2.5 Puncturable Pseudorandom Function . . . 16
- 2.6 Non-interactive Zero Knowledge . . . 17
- 2.7 Digital Signatures . . . 18

3 Mergeable Arguments **19**
- 3.1 Trapdoor Extraction . . . 21

4 Mergeable Arguments from BARGs **22**
- 4.1 Construction . . . 22
- 4.2 Merge Efficiency for Circuits . . . 26
  - 4.2.1 Definition. . . . 26
  - 4.2.2 One-Hop Argument System. . . . 27
  - 4.2.3 Construction. . . . 28
  - 4.2.4 Analysis. . . . 30
- 4.3 Universal Soundness . . . 32

5 Mergeable Arguments from Indistinguishability Obfuscation **32**
- 5.1 Parameters . . . 33
- 5.2 Construction . . . 34
- 5.3 Soundness . . . 36

6 Application I: CCA Secure Fully Homomorphic Encryption **44**
- 6.1 Trapdoor Languages for CCA1 FHE . . . 45
- 6.2 Construction . . . 46
- 6.3 Analysis . . . 47

7 Application II: Adaptive Multi-Hop Aggregate Signatures **49**
- 7.1 Definition . . . 49
- 7.2 Construction . . . 51
- 7.3 Analysis . . . 53

A Universal SNARG for Mergeable Trapdoor Languages **55**
A.1 Predicate Extractable Commitments with Prefix Consistency 55
A.2 Construction 59
A.3 Analysis 61

1 Introduction

SNARGs are typically studied in the CRS model and their soundness is required to hold only against cheating provers that are computationally bounded. Given a witness w, the prover should run in time $\mathrm{poly}(T)$ where T is the time required to verify w.[1] The proof size should be sublinear in w and the verification time should be sublinear in T.

In this work we study *mergeable arguments* that allow proofs for simple statements to be merged into proofs for more complex statements. Consider a language L equipped with an efficient *instance-merging* procedure that takes a sequence of instances $x_1, \ldots, x_k$ and produces a new instance x such that if $x_1, \ldots, x_k \in L$ then also $x \in L$. Mergeable arguments for L feature a *proof-merging* procedure which, given instances $x_1, \ldots, x_k$, together with an accepting proof for each instance, generates an accepting proof for the merged instance x. We require that the merged proof is shorter than the combined length of the original proofs.[2] Since the proof merger is required to work given any accepting proofs, even ones produced by the merger itself, instances and proofs can be merged recursively.

Starting from the work of [Val08], the idea of proof merging has been studied extensively for the purpose of verifying distributed computations. A distributed computation is executed via a multi-party protocol, where in each round, each party combines its local input with its incoming messages from other parties to produce its output and/or outgoing messages. To make the computation verifiable, we augment each message with a proof of its validity. Given the validity proofs of each of its incoming messages, a party merges these proofs into a validity proof for its output and outgoing messages.

Previous work (e.g. [Val08, CT10, BCMS20, CHL+25]) introduced notions of mergeable arguments that are closely related to the one studied in this work and suggested constructions based on the idea of recursive composition of *succinct non-interactive arguments of knowledge for NP* (SNARKs). Given SNARK proofs for instances $x_1, \ldots, x_k$ they generate a proof for the merged instance x by constructing a SNARK proof for the following NP statement: there exist instances $x_1, \ldots, x_k$ and proofs $\pi_1, \ldots, \pi_k$ such that (i) for each i, π_i is an accepting proof for x_i, and (ii), x is the result of merging the instances $x_1, \ldots, x_k$. To argue soundness, they recursively invoke the SNARK's knowledge extractor and reconstruct a *merge tree* for x: a tree where each node is labeled by an instance, and the label of each internal node is given by merging the instances on its children. The root label of the extracted tree is x and, they additionally extract a valid witness for the instance in each leaf. Since L is closed under merging, it follows that $x \in L$.

The knowledge soundness of existing SNARK candidates is typically analyzed in idealized models. In the plain model, the analysis of these constructions relies on heuristics or non-falsifiable knowledge assumptions and does not include an explicit knowledge extractor, the existence of which is subject to strong barriers [BCPR16, BP15]. Moreover, since the security analysis of mergeable arguments is based on recursive knowledge extraction, and since extraction is typically assumed to have polynomial overhead, such constructions can only support a constant depth for the merge tree. To go beyond constant depth, several works relied on a stronger assumption: the existence of a SNARK with a so-called straight-line extractor that only introduced an additive overhead [CT10, CGSY24, CHL+25].

The maximal tree barrier. Even SNARKs with straight-line extraction, however, are not known to imply mergeable arguments that allow for arbitrary recursive merging. For an instance x, we denote by $\omega(x)$ the

[1] In the introduction we ignore polynomial factors in the security parameter.

[2] SNARGs for deterministic languages have a trivial proof merger that ignores the proofs for $x_1, \ldots, x_k$ and generates a proof for x from scratch. Therefore, in this setting we also require that proof merging requires less time than proving the statement from scratch.

size of the maximal merge tree for x. Since the analysis described above involves extraction of the entire merge tree, soundness is only argued to hold for instances x where $\omega(x)$ is polynomially bounded. This, in particular, excludes instances x resulting from merging statements following a directed acyclic graph (DAG) such that the merge tree obtained by "unrolling" the DAG (duplicating nodes with fan-out > 1) is of super-polynomial size. In this work we refer to this technical hurdle as the *"maximal tree"* barrier.

As far as we know, the only construction of mergeable arguments that surpasses the maximal tree barrier was recently proposed by Chakraborty et al. [CHL+25]. Their solution is based on SNARKs with straight-line extraction and on a new primitive that they call adversarial one-way functions. Very roughly, an adversarial one-way function is a hash function such that no adversary can produce an image y in time t and later find a hash tree rooted at y with $T \gg t$ distinct leaves. By adding a hash evaluation into the proof merging procedure, they guarantee that the extraction process always reconstructs a merge tree with a polynomial number of distinct leaves. Chakraborty et al. show that adversarial one-way functions exist in the random oracle model. In the plain model, however, no constructions are known based on standard assumptions.

Recently, [PP26] studied mergeable arguments for deterministic languages and gave a construction under the LWE assumption without relying on SNARKs. The analysis of [PP26] differs from that of previous constructions for non-deterministic languages: Instead of extracting the entire merge tree, they only extract a single random path in the tree. Nonetheless, their construction encounters the same barrier since the the reduction's success probability degrades with $\omega(x)$. Even when restricting to deterministic computations, it is not known how to overcome the maximal tree barrier without the assumptions of Chakraborty et al. [CHL+25].

Motivated by the above discussion we ask:

Can we construct mergeable arguments for useful languages
that avoid the maximal tree barrier based on standard assumptions?

A motivating application: CCA1 homomorphic encryption. To demonstrate the significance of mergeable arguments that can surpass the maximal tree barrier, we highlight one use-case addressed in this work: constructing multi-hop fully homomorphic encryption (FHE) with CCA1 security. Recall that [NY89] proposed a simple transformation making any encryption scheme CCA1 secure, utilizing the double-encryption paradigm of [DDN03]: encrypt each message under two public keys and provide a NIZK proof that both ciphertexts encrypt the same message. We can directly apply this transformation to any FHE scheme, and what remains is to specify how to perform homomorphic evaluation. While we can homomorphically compute under each public key, it is not clear how to construct a proof for the fact that the two evaluated ciphertexts still encrypt the same message.

The notion of mergeable arguments suggests a natural solution: let L_{CCA} be the language that contains pairs of FHE ciphertexts encrypting the same message under the two public keys. Consider the instance merger that given two instances in L_{CCA} performs the same homomorphic operation under each key, resulting in a new instance in L_{CCA}. Using mergeable arguments for the aforementioned instance merger, we can merge the proofs of the two instances to obtain a proof for the new one.[3] To allow homomorphic evaluation of arbitrary polynomial size circuits, however, requires mergeable arguments beyond the maximal tree barrier. For example, if we homomorphically evaluate λ iterated squaring operations, then the resulting ciphertext has a merge tree of size 2^{λ}.

[3] For the scheme to be secure the mergeable arguments must hide the witnesses for the merged statements. This can be achieved by composing mergeable arguments with a standard NIZK.

1.1 Our Results

In this work we construct mergeable arguments for the class of *trapdoor NP languages* [CGKS23, DJJ+25]. We give two constructions: The first construction is based on LWE, and its proof size grows with the depth of the merge tree and with the size of a single witness. The second construction is based on iO and its proof size grows only with the security parameter. In both constructions, security degrades with the depth of the maximal merge tree, but not with its size. We demonstrate two applications of our mergeable arguments for trapdoor languages: CCA1-secure multi-hop FHE and strong multi-hop aggregate signatures.

Trapdoor languages. A trapdoor language is a keyed family of NP languages $L = \{L_\beta\}$ where one can sample a key β together with a trapdoor τ that can be used to decide membership in L_β in polynomial time. For example, the language L_{CCA} described above is a trapdoor language: the key β is the pair of public keys and the trapdoor contains the corresponding secret keys. Using the trapdoor we can check if a given pair of ciphertexts encrypt the same message or not in polynomial time.

We note that in order to generate, merge and verify proofs, only the key β is needed, and the trapdoor is only used in the security reduction. However, the time required to generate and merge proofs does grow with the time to decide L_β using the trapdoor. This should be contrasted with the mergeable arguments of [PP26] for deterministic languages where the proof generation and merge time are independent of the time to decide the instance.

Constructions. Our first construction is based on batch arguments satisfying specific properties that can be instantiated under LWE.

Theorem 1.1 (Informal)**.** *Let L be a trapdoor language with an instance merging procedure. Assuming hardness of LWE, there exist mergeable arguments for L. For security parameter λ, the CRS size is* $\mathrm{poly}(\lambda)$*. For instance x with witness w the proof size is $|w|$. Given instances $x_1, \ldots, x_k$ and their proofs $\pi_1, \ldots, \pi_k$, the merged proof is of size*

$$\left(\sum_i |x_i|\right) \cdot \mathrm{poly}(\lambda) + \max_i |\pi_i| \ .$$

Note that the length of our merged proof grows with the length of a single proof π_i (and not with the sum of the lengths). This is non-trivial succinctness, similar in nature to what is achieved by batch arguments.

We note that Theorem 1.1 imposes no a priori bound on the depth of the merge tree. If the number of instances k and size of each instance are bounded by some fixed polynomial in λ, then after d recursive merges, the proof size is $d \cdot \mathrm{poly}(\lambda)$. The construction also satisfies a notion of *knowledge extraction* that will be useful in our applications.

We give an extension of the above construction that reduces the proof size in the setting where multiple merge operations are performed by the same party in a single "hop". Consider a party merging instances and their proofs following some DAG of depth d and in-degree c. If it uses the proof merger given by the theorem above, the additive overhead in proof size will be $d \cdot c \cdot \mathrm{poly}(\lambda)$. By performing all merges together, we can reduce this overhead to be independent of d.

Theorem 1.2 (Informal)**.** *Let L be a trapdoor language with an instance merging procedure. Assuming hardness of LWE, there exist mergeable arguments for L with CRS and proof size as in Theorem 1.1. For security parameter λ, instances $x_1, \ldots, x_k$, their proofs $\pi_1, \ldots, \pi_k$, and a DAG of in-degree c, the merged proof according to the DAG is of size*

$$c \cdot \max_i |x_i| \cdot \mathrm{poly}(\lambda) + \max_i |\pi_i| \ .$$

Our second construction is based on indistinguishability obfuscation (iO) with sub-exponential security and it achieves proofs whose size does not grow with the merge tree depth.

Theorem 1.3 (Informal)**.** *Let L be a trapdoor language with an instance merging procedure. Assuming the existence of iO and OWF with sub-exponential security, there exist mergeable arguments for L. Both the CRS and the proof have size* $\mathsf{poly}(\lambda)$.

CCA1 FHE. As an application of our mergeable arguments for trapdoor languages we give the first construction of CCA1-secure multi-hop FHE based on standard assumptions.

Theorem 1.4 (Informal)**.** *Assuming hardness of LWE and the existence of fully (resp. leveled) multi-hop homomorphic encryption, there exists a CCA1-secure fully (resp. leveled) multi-hop homomorphic encryption scheme in which the ciphertext size grows linearly with the number of hops.*

Theorem 1.5 (Informal)**.** *Assuming the existence of iO and OWF with sub-exponential security, there exists a CCA1-secure multi-hop fully homomorphic encryption scheme in which ciphertext size is independent of the number of hops.*

Following the proofs of Theorems 1.4 and 1.5, we can use mergeable arguments to get FHE with a more general property that is related to the notion of *targeted malleability* [BSW12]. Let A be a property of ciphertexts that can be tested efficiently given the secret key. Let B be a set of homomorphic operations that preserve the property A. Then we can construct a multi-hop FHE scheme where ciphertexts satisfying the property A contain a publicly verifiable proof of this fact.

We briefly mention previous work on CCA1 FHE. Based on SNARKs, the work of [CRRV17] gave a *single-hop* CCA1 FHE.[4] In single-hop FHE we can homomorphically evaluate over freshly encrypted ciphertexts, but once ciphertexts have undergone homomorphic evaluation, we cannot compute on them further. While a single-hop to multi-hop transformation for CPA-secure FHE is straightforward using bootstrapping [GHV10], we observe that it fails to provide CCA1 security.[5]

The work of [BSW12] used a stronger notion of SNARKs with a linear time extractor to construct CCA1 FHE that supports *layered* multi-hop evaluation: in each hop we can compute only the ciphertext resulting from the previous hop, and the evaluation must be performed by a single party operating on all these ciphertexts together. However, we cannot compute jointly on ciphertext evaluated by two different parties. The work of [CHL+25] constructed multi-hop CCA1 FHE based on SNARKs with straight-line extraction and adversarial one-way functions.

Recently, Yang, Yu and Susilo [YYS25] constructed a single-hop CCA1 FHE based on LWE avoiding SNARKs altogether. One limitation of their scheme is that they need to set an a priori bound S on the size of the evaluated circuit and the running time of their security reduction grows with the bound S. On the one hand, setting S to a specific polynomial limits the homomorphic capabilities of the scheme. On the other hand, setting S to be super-polynomial requires assuming super-polynomial security of LWE. In contrast, in our multi-hop scheme there is no bound on the size of each hop.

Multi-hop aggregate signatures. Our second application improves the existing constructions of multi-hop aggregate signatures. In aggregate signatures, a set of signatures generated by different parties can

[4] The same work also proposes other single-hop CCA1 FHE candidates without the double encryption paradigm. From LWE, where the ciphertext size grows with the number of inputs to the circuit, and from iO and Lossy Encryption.

[5] The transformation involves providing an encryption of the secret key, which can be decrypted by the decryption oracle in the CCA1 security game.

be combined into one short signature that can be verified given the set of messages signed and the set of verification keys they were signed under. Recently [DGKV22] constructed a multi-hop aggregate signature scheme, where aggregated signatures can be further aggregated. The signature length in their construction grows linearly with the depth of recursive aggregation. However, the construction of [DGKV22] has some notable limitations:

- There is an a priori bound on the aggregation tree depth, and the CRS size grows with this bound.
- Verification requires the complete structure of the aggregation tree.
- Security requires the aggregation tree to be of polynomial size and fixed statically at CRS generation.

Based on our mergeable arguments with knowledge extraction, we construct an aggregate signature scheme that overcomes these limitations:

- The CRS is short, and there is no a priori bound on aggregation tree depth.
- Verifying aggregate signatures requires only a short *digest* of the message–key pairs (essentially a hash tree). In particular, verification does not need to know the aggregation tree and can be sublinear in the number of messages signed.
- Security excludes any efficiently computable forgery. Moreover, we strengthen security to disallow more general forgeries: instead of producing the full aggregation tree, a forgery consists of a local opening of just one of the message–key pairs not queried by the signing oracle.

In particular, our scheme allows for aggregation strategies following any DAG, even if the size of the unrolled aggregation tree is super-polynomial.

Theorem 1.6. *Assuming hardness of LWE, there exist multi-hop aggregate signatures with the properties above in which the signature size grows linearly with the number of hops.*

Our construction can be based on any plain signature scheme and aggregate it without modifying its key generation or signing algorithms.

Incrementally verifiable computation with universal soundness. For our last application we give a new construction of incrementally verifiable computation (IVC) for trapdoor languages that achieves a stronger notion of *universal soundness* [BG08]. In IVC, we would like to prove the correctness of a computation as it progresses. The computation starts from some initial configuration $x_0 \in \{0,1\}^n$ and in each step the configuration is updated following a procedure S running in time $\mathrm{poly}(n)$. Given the configuration x_i at time i and some auxiliary input z_i for the step, the configuration at time $i+1$ is given by $x_{i+1} = S(x_i, z_i)$. We can define the language L^* of *reachable configurations* such that $(x_0, x_i, i) \in L^*$ if and only if:

- $i = 0$ and $x_0 = x_i$, or,
- there exists $(x_0, x_{i-1}, i-1) \in L^*$ and z_{i-1} such that $x_i = S(x_{i-1}, z_{i-1})$.

An IVC scheme is a SNARG for L^* where proofs can be *efficiently updated*. Given a proof π_i for $(x_0, x_i, i) \in L^*$ and any z_i, we can obtain a proof π_{i+1} for $(x_0, x_{i+1}, i+1) \in L^*$ where $x_{i+1} = S(x_i, z_i)$. Importantly, the time to update the proof should only grow with the running time of S and not with the number of steps i.

The works of [DGKV22, PP22] constructed IVC for P under LWE. In IVC for P, the auxiliary input z_i is always empty (therefore, we can decide if $(x_0, x_i, i) \in L^*$ deterministically in time linear in i). IVC for general NP languages can be constructed based on SNARKs for NP [Val08]. More recently [DJJ+25] gave a construction under LWE and iO.

The standard notion of soundness for IVC only considers statements (x_0, x_i, i) where i is polynomial. That is, we require that for any adversary, polynomial P and security parameter λ, the adversary given an honestly generated CRS can output a statement $(x_0, x_i, i) \notin L^*$ such that $i \leq P(\lambda)$ together with an accepting proof with probability at most $\mathsf{negl}(\lambda)$. The notion of *universal soundness* strengthens the above requirement by allowing the adversary to output any statement $(x_0, x_i, i) \notin L^*$ without the restriction $i \leq P(\lambda)$. Universal soundness is required in applications where the statement (x_0, x_i, i) is adversarially chosen and there is no a priori polynomial bound on i.

In all existing IVC schemes, given an adversary that outputs a proof for a false statement $(x_0, x_i, i) \notin L^*$, the running time of the soundness reduction grows with i. Therefore, we cannot show that these schemes satisfy universal soundness based on polynomial-time hardness assumptions.

IVC with universal soundness can be based on hardness assumptions against $\mathrm{poly}(B)$-time adversaries where $B(\lambda)$ is super-polynomial. However, this requires that we restrict the completeness of the scheme to statements $(x_0, x_i, i) \in L^*$ where $i \leq B(\lambda)$.[6] Finally, we mention the work of [BPSS23] that constructed IVC with universal soundness without relying on super-polynomial hardness assumptions. Instead, they rely on a worst-case complexity-theoretic assumption related to fine-grained cryptographic puzzles.

We give a construction of IVC for trapdoor languages L^* assuming polynomial hardness of LWE. Our construction is the first to achieve universal soundness, without relying on super-polynomial assumptions or cryptographic puzzles.

Theorem 1.7 (Informal.). *Assuming LWE, there exists an IVC scheme with universal soundness for any trapdoor language L^*. The proof length for a statement $(x_0, x_i, i) \in L^*$ is $\mathrm{poly}(\lambda, n, m, \log i)$ where λ is the security parameter, n is the size of the configuration and m is the size of the auxiliary input.*

1.2 Technical Overview

We start by describing our notion of mergeable arguments in more detail. Our notion is adapted from [PP26] to the setting of trapdoor languages. Let L be an NP language equipped with a polynomial-time *instance-merging procedure* IM that, given instances $x_1, \ldots, x_k$, outputs a merged instance x or a special symbol $\perp$ indicating that $x_1, \ldots, x_k$ cannot be merged. In some applications it will be useful to consider L that is not closed under instance merging. In this setting, we denote by L^* the closure of L under IM. That is, L^* is the smallest set that contains L such that if $x_1, \ldots, x_k \in L^*$ and $\mathsf{IM}(x_1, \ldots, x_k)$ outputs $x \neq \perp$, then $x \in L^*$. For example, consider the language L of all fresh FHE ciphertexts under some public key, with the instance merger that takes two ciphertexts and outputs their homomorphic addition or multiplication. In this case, L^* contains all evaluated ciphertexts.

Our mergeable argument for L includes, in addition to the standard CRS generation and prover/verifier algorithms, a polynomial-time *proof-merging procedure*. Intuitively, for any set of instances that can be successfully merged, the proof merger should be able to merge their proofs. More formally, given instances $x_1, \ldots, x_k$ and proofs $\pi_1, \ldots, \pi_k$, the proof merger PM outputs a merged proof π. We require that if each π_i is an accepting proof for x_i and $\mathsf{IM}(x_1, \ldots, x_k)$ outputs $x \neq \perp$, then π must be an accepting proof for x.

[6] Since the time to prove $(x_0, x_i, i) \in L^*$ honestly grows with i, most application only requires completeness for polynomial i. Some applications, however, such showing hardness in the class PPAD, require completeness for i that is super-polynomial [KPY20].

If L is not closed under IM, then we cannot hope to obtain soundness for L (since any output of IM should have an efficient valid proof) and the best we can hope for is soundness with respect to the closure L^*. That is, no efficient adversary should be able to produce $x \notin L^*$ together with an accepting proof with non-negligible probability. In this work, we achieve a weaker notion of soundness: for every language $D \in P$ such that $L^* \subseteq D$, we guarantee soundness with respect to D. At first glance, this notion may appear trivial since one could simply prove membership in D using a standard SNARG for P. What makes it useful is that the language D only need to be set in the analysis and, in particular, its description is not needed to generate, merge or verify proofs. Now, by considering a family of languages $L = \{L_\beta\}$ and instance mergers $\mathsf{IM} = \{\mathsf{IM}_\beta\}$, we can sample β such that L_β is a hard language, but for each β there exists $L^*_\beta \subseteq D_\beta \in P$. We refer to the decision procedure for D_β as the *trapdoor* for L_β. For example, if L_β is the language of all fresh FHE ciphertexts under some public key β, and L^*_β contains all evaluated ciphertexts, then we can take D_β to be the set of all decryptable ciphertexts, which can be decided using the secret key corresponding to β.

Extraction. We also consider a notion of knowledge extraction which generalizes soundness. Like soundness, extraction is also defined with respect to a language $D \in P$, except that instead of requiring that $L^* \subseteq D$ we only require that D is closed under merging (i.e. D may not contain L). If an adversary produces an instance $x \notin D$ together with an accepting proof, it must be that at least one of the leaves in the merge tree of x is labeled by some instance $x' \notin D$. The extraction property states that (i) we can sample a CRS that is "programmed" on D together with an extraction trapdoor td such that the programmed CRS is indistinguishable from an honestly generated one, and (ii) there exists an extractor such that whenever an efficient adversary outputs an instance $x \notin D$, together with an accepting proof π for x under the programmed CRS, the extractor, given td, x and π outputs an instance $x' \in L \setminus D$ with a corresponding witness for $x' \in L$.

This notion indeed generalizes soundness: if an adversary outputs an accepting proof π for $x \notin D$ with non-negligible probability then this also happens when the CRS is programmed on D. However, if $L \setminus D = \emptyset$ we reach a contradiction.

1.2.1 Mergeable arguments from LWE.

Our main technical contribution is the construction of mergeable arguments for trapdoor languages from LWE that goes beyond the maximal tree barrier. We describe our construction below.

To generate a fresh proof for $x \in L$, we simply use the NP witness. In applications that require zero-knowledge we instead use a NIZK proof. To merge proofs we use a *batch argument (BARG)*, a non-interactive argument for a conjunction of NP statements whose proof size grows with the size of a single witness and not with the number of statements. The proof merger PM is given instances $x_1, \ldots, x_k$ and proofs $\pi_1, \ldots, \pi_k$. PM outputs a merged proof π that contains the original instances $x_1, \ldots, x_k$ and a BARG proving that for every i, there exists an accepting proof π_i for x_i. To verify the merged proof for the statement x, simply check that $\mathsf{IM}(x_1, \ldots, x_k) = x$ and verify the BARG. To support an unbounded number of recursive merges the BARG must be very efficient: for a witness of size m, the CRS should not grow with m and the proof should be of size $m + \mathsf{poly}(\lambda)$. Such BARGs were recently constructed in [DMP26].

Our reduction takes a merged proof π for a statement $x \notin D$, and extracts an accepting proof for some leaf x' in the merge tree such that $x' \notin D$ together with a witness for $x' \in L$. Unlike SNARKs for NP, the BARG extractability property does not allow to recursively extract the entire merge tree. Instead, we can extract from every accepting BARG proof, a witness for one of the statements in the conjunction. The index of the statement to extract is "programmed" into the CRS in a way that is oblivious to the adversary. In our

setting, given an accepting proof for some node y in the merge tree consisting of instances $y_1, \ldots, y_k$ we can extract an accepting proof π_i for some y_i at the level below. Therefore, we can recursively extract a complete path in the merge tree.

The challenge is how to guarantee that we extract a path to a leaf $x' \notin D$. Using techniques from [DGKV22, PP22] there is a solution when the depth of the merge tree is logarithmic. By programming the CRS to be extractable along a random path in the tree we can extract a leaf $x' \notin D$ with non-negligible probability. In a recent work, [PP26] showed how to extend this method to *any* merge tree of polynomial-size (regardless of its depth). They do this by having the proof in each node carry the tree sizes associated with all of its children. In the reduction, they sample one of the children to extract from based on these weights. The probability of extracting a leaf $x' \notin D$ is now inversely proportional to the size of the maximal merge tree for x.

To implement this idea, [PP26] used BARGs with a stronger *functional extraction* property, where instead of programming the extraction path in the CRS, the path is computed in a step-by-step manner as a function of the aforementioned weights. In more detail, we can generate a CRS that is "programmed" on a function f together with an extraction trapdoor td such that the programmed CRS is indistinguishable from an honestly generated CRS (in particular, the CRS hides f). During proof generation the prover can choose a *tag* t and both proof generation and verification will use this tag. The extraction guarantee is that using the trapdoor td we can extract the witness for statement $i = f(t)$ from any accepting proof with all but negligible probability. BARGs with functional extraction and the required rate can be constructed from LWE [BCJP24, DMP26].

In this work, we show how to use functional extraction to overcome the maximal tree barrier for trapdoor languages. Our tag t contains the statements $(x_1, \ldots, x_k)$, and in the analysis, we program the CRS on a function f that contains the trapdoor of L (which is the decision procedure of D) hard-coded. The function $f(t)$ uses the trapdoor to output the first index i such that $x_i \notin D$. Therefore, given $x \notin D$ our extraction procedure necessarily follows a path of instances outside D all the way to a leaf $x' \notin L \subseteq D$, regardless of the size of the merge tree. This is in contrast to previous work that extracted a random leaf and therefore suffered a loss proportional the the size of the merge tree.

It remains to argue that when replacing the honest CRS with the programmed CRS the probability that the adversary outputs an instance $x \notin D$ together with an accepting proof can only change by a negligible amount. This is where we use the CRS indistinguishability which holds even given D's decision procedure.

Finally, we argue that the reduction is efficient. The running time of the extraction scales with the depth of the merge tree. We therefore require that each proof contains the depth of its merge tree in unary representation. Thus, the reduction always runs in polynomial time in the length of the proof produced by the adversary.

Improved Efficiency for Large Hops. In the mergeable arguments described above, the proof size grows with each individual merge operation. We now show how to avoid this growth in settings where a single party performs multiple merge operations within a single "hop." For example, in the homomorphic encryption application mentioned above, each party may homomorphically evaluate a large circuit C by performing a merge for each gate. We show that, given a sequence of merges each with bounded fan-in c, the entire sequence can be performed together so that the increase in proof size scales only with the fan-in c, and not with the number of merges.

Our strategy is to combine our multi-hop recursive extraction technique with a separate proof system for proving the correctness of each hop. In this one-hop system, the proof for each hop is of fixed polynomial size that depends only on the fan-in c of C and not on the number of merges. Our one-hop proof system is

based on BARGs, together with *predicate-extractable hash* [BBK+23, YYS25]. In a predicate-extractable hash, we generate a key that is programmed on a secret predicate f, together with an extraction trapdoor. We can then hash a vector of values $\vec{x} = (x_1, \ldots, x_N)$ into a short digest that supports local opening of any x_i. Given a digest and the trapdoor, we can extract the index of *some* value satisfying the predicate f (or $\perp$ if no such index exists). We consider predicate-extractable hash satisfying a stronger extraction property that we call *prefix consistency* where we are guaranteed to extract the index of the *first* value satisfying the predicate f. The correctness guarantee is that if the extracted index is i, it is hard to open to x_i such that $f(x_i) = 0$ and for any $j < i$ (or, if $i = \perp$ for any $j \leq N$), it is hard to open to x_j such that $f(x_j) = 1$.

The one-hop proof system follows the so-called *hash and BARG* paradigm [CJJ22, KVZ21] where the hash used is predicate-extractable. When merging instances $(x_1, \ldots, x_k)$ according to a merge circuit C, every wire carries a merged instance. We compute a hash of the instances $\vec{x} = (x_1, \ldots, x_N)$ along the wires of C in some topological order. We additionally compute a BARG proof, where the i-th statement argues that the hash can be opened to x_i and $x_{i_1}, \ldots, x_{i_c}$ such that C computes x_i by merging $x_{i_1}, \ldots, x_{i_c}$ and the values opened are consistent with IM. We program the hash key on the predicate f that has the language's trapdoor hard-coded and outputs $f(x) = 1$ if $x \notin D$.

We claim that if the output $x_N \notin D$, then the extracted index i has to be an index of an input instance (i.e. $i \in [k]$) such that $x_i \notin D$. Indeed, if $k < i \leq N$ is the output wire of some merge gate, and we program the BARG on i, then we can extract instances $x_{i_1}, \ldots, x_{i_c}, x_i$ together with local openings of the hash, such that $IM(x_{i_1}, \ldots, x_{i_c}) = x_i$. However, extraction with prefix consistency guarantees that $x_i \notin D$ while $x_{i_1}, \ldots, x_{i_c} \in D$, in contradiction to the instance merger soundness. Otherwise, if $i = \perp$, we program the BARG on N and get that $x_N \in D$ in contradiction.

Note that to reach a contradiction we must program the BARG on the same index i extracted from the hash. While the index i is not yet fixed when the BARG CRS is sampled, we can simply program the BARG on a random i. This, however, introduces a security loss that grows with $|C|$. Previous work [BBK+23, YYS25] dealt with this loss by fixing an a priori bound on $|C|$. To support hops of unbounded size, we take a different route. We, again, rely on functional BARGs and program the BARG trapdoor on a function that has the hash trapdoor hard-coded and given the digest as a tag, extracts the correct index i.

Finally, we use this single-hop construction to get a multi-hop construction that supports large hops as follows. As in our original multi-hop construction above, the merged proof for $(x_1, \pi_1), \ldots, (x_k, \pi_k)$ consists of the instances $x_1, \ldots, x_k$ and a BARG claiming that each x_i has an accepting proof. We also add a one-hop proof that $x_1, \ldots, x_k$ are merged correctly according to C. This results in a merged proof that does not grow with $|C|$ but does grow with the number of inputs k. To get a proof that grows only with the fan-in c, rather than including the instances $x_1, \ldots, x_k$ we again use the predicate-extractable hash to compute a digest of $x_1, \ldots, x_k$. We also add the authentication path for each x_i to the witness for the i-th BARG statement. Recall, however, that in the analysis of the original construction above, we programmed the BARG on a function that contains the language's trapdoor and given $x_1, \ldots, x_k$ as the tag, extracts the first index i such that $x_i \notin D$. Since the proof no longer contains $x_1, \ldots, x_k$ in the clear, we instead (i) program the predicate-extractable hash on a function that contains the language's trapdoor and extracts an index of some $x_i \notin D$, and (ii) program the BARG on a function that contains hash trapdoor and extracts the index i given the digest as the tag. It remains to check the consistency of digest in the proof and the digest contained inside the single-hop proof. To this end we simply use the same digest for both constructions (where for the multi-hop proof we only care about the first k indices).

1.2.2 Mergeable arguments from iO.

In our LWE construction above, the proof size grows with the depth of the merge tree. Next, we describe a construction based on sub-exponential iO, in which the proof does not increase at all with each merge. Our starting point is the incrementally verifiable computation (IVC) construction for trapdoor languages from [DJJ+25]. In IVC the goal is to construct a SNARG for a computation incrementally, while the computation is taking place. Assuming sub-exponential IO, [DJJ+25] construct IVC for trapdoor language where the proof size does not grow with the size of the computation's state. We show that a simple adaptation of their construction gives mergeable arguments with short proofs that overcomes the maximal DAG barrier. Next we recall the IVC construction of [DJJ+25], and explain our changes.

Given a fixed circuit C, the IVC language L_{IVC} consists of statements of the form (x_0, x_i, i) if there exist witnesses $w_1, \ldots w_i$ such that $C(x_{j-1}, w_j) = x_j$ for all $j \leq i$. The CRS consists of obfuscated programs for the prover and verifier, each hard-coded with a puncturable PRF key K. The prover's obfuscated program, given an instance $(x_0, x_{i-1}, i-1)$, proof π_{i-1} and witness w_i proceeds as follows:

- Verify that $\mathsf{PRF}_K(x_0, x_{i-1}, i-1) = \pi_{i-1}$. Abort if verification fails.
- Compute the next instance $x_i = C(x_{i-1}, w_i)$.
- Compute the next proof $\pi_i = \mathsf{PRF}_K(x_0, x_i, i)$.
- Output $((x_0, x_i, i), \pi_i)$.

The verifier's obfuscated program, given an instance (x_0, x_i, i) and proof π_i, checks that $\pi_i = \mathsf{PRF}_K(x_0, x_i, i)$.

The high-level structure of the security proof is by induction on the number of steps i. For each step i we assume that the all the proofs for false statement of the form $(x_0, x_{i-1}, i-1) \notin L_{\mathsf{IVC}}$ have been removed by puncturing the PRF. Observe that:

$$C(x_{i-1}, w_i) = x_i \ \wedge \ (x_0, x_i, i) \notin L_{\mathsf{IVC}} \quad \rightarrow \quad (x_0, x_{i-1}, i-1) \notin L_{\mathsf{IVC}} \ .$$

Therefore, since the proof for each instance $(x_0, x_{i-1}, i-1) \notin L_{\mathsf{IVC}}$ was removed, the prover's program never outputs a proof for $(x_0, x_i, i) \notin L_{\mathsf{IVC}}$. We can now puncture out the proofs for every false statement $(x_0, x_i, i) \notin L_{\mathsf{IVC}}$ in an exponential sequence of hybrids. Here we crucially use the trapdoor to recognize L_{IVC} efficiently inside the prover's program.

While the scheme of [DJJ+25] updates the proof for one instance at a time, we observe that by merging multiple proofs into one we get mergeable arguments. In more detail, to get mergeable arguments for a language L with instance merger IM we modify the scheme as follows: First, we replace the circuit C updating a single instance, with the instance merger. We also augment each instance x with the depth d of its merge tree so that merging the augmented instances $(x_0, d_0), \ldots, (x_k, d_k)$ gives the instance (x, d) such that $\mathsf{IM}(x_0, \ldots x_k) = x$ and $d = \max_j d_j + 1$. In the security proof, we use the depth d in the same way that [DJJ+25] use the number of steps i: For each d we assume that the all the proofs for false statement of the form (x, d') with $x \notin L$ and $d' < d$ have been removed by puncturing the PRF. Observe that:

$$\mathsf{IM}((x_0, d_0), \ldots, (x_k, d_k)) = (x, d) \ \wedge \ x \notin L \quad \rightarrow \quad j \in [k], \ x_j \notin L \ \wedge \ d_j < d \ .$$

Therefore, since the proof for each instance (x, d') with $x \notin L$ and $d' < d$ was removed, the prover's program never outputs a proof for (x, d) with $x \notin L$. We can now puncture out the proofs for every false statement (x, d) with $x \notin L$ in an exponential sequence of hybrids.

1.2.3 CCA1 FHE from mergeable arguments.

Our first application is to multi-hop FHE with CCA1 security. Our construction follows the [NY89,CRRV17] paradigm. Starting from any FHE scheme we modify it so that the public key pk contains two public keys $\mathsf{pk}_0, \mathsf{pk}_1$ and a NIZK CRS. To encrypt a bit, we encrypt it under the two keys and compute a NIZK proof that the two ciphertexts are indeed fresh encryptions of the same bit. Homomorphic evaluation is then done by evaluating on each ciphertext, and merging the accompanying proofs using our mergeable arguments. In more detail, we use mergeable arguments for the language L_β parameterized by $\beta = \mathsf{pk}$ which contains all pairs of fresh ciphertexts encrypting the the same bit under $\mathsf{pk}_0, \mathsf{pk}_1$, with the NIZK proof serving as the witness. The instance merger performs the same homomorphic operation under each key. We use the corresponding proof merger during homomorphic evaluation.

The soundness guarantee we get is with respect to the natural trapdoor language $D \in P$ that contains all ciphertext pairs $\mathsf{ct}_0, \mathsf{ct}_1$ such that $\mathsf{Dec}(\mathsf{sk}_0, \mathsf{ct}_0) = \mathsf{Dec}(\mathsf{sk}_1, \mathsf{ct}_1)$. Indeed, this soundness guarantee is enough to argue CCA security, following the analysis of [NY89]. To get soundness for D we need to guarantee that indeed $L^* \subseteq D$ where L^* contains all evaluated ciphers that result from merging. For this condition to hold, we need the NIZK to be statistically sound. That is, we need that with high probability over the NIZK CRS, there does not exist a valid witness for a statement outside of D). We note that a computationally sound NIZK can be used if the mergeable arguments satisfy the stronger notion of knowledge extraction.

1.2.4 Strong multi-hop aggregate signatures.

Our second application is to aggregate signatures. Our starting point is the construction of (single-hop) aggregate signatures from BARGs in [WW22]. We start with some base signature scheme with verification algorithm Verify. Given a sequence $\{\mathsf{vk}_i, \mathsf{m}_i, \sigma_i\}_{i\in[k]}$ such that σ_i is a signature on m_i under vk_i, the aggregate signature is a BARG proof that for each $i \in [k]$, there exists σ_i such that $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{m}_i, \sigma_i) = 1$.

In the aggregate unforgeability game an adversary receives some challenge verification key vk^* and access to a signing oracle $\mathsf{Sign}(\mathsf{sk}^*, \cdot)$, and outputs an aggregate signature for a sequence of key-message pairs $\{(\mathsf{vk}_i, \mathsf{m}_i)\}_{i\in[k]}$ containing $(\mathsf{vk}^*, \mathsf{m}^*)$ such that the signing oracle was not queried on m^*. To reduce to the unforgeability of the base signature scheme we need to extract a valid signature σ^* for $(\mathsf{vk}^*, \mathsf{m}^*)$. [WW22] achieve this by programming the BARG CRS on a random index $i \in [k]$. By the CRS indistinguishability of the BARG, $(\mathsf{vk}_i, \mathsf{m}_i) = (\mathsf{vk}^*, \mathsf{m}^*)$ with probability close to $1/k$. By the BARG's extraction property we can extract a valid signature. This gives security for any polynomially bounded k.

The original construction of [WW22] was limited to a single hop of aggregation. The work of [DGKV22] extended this approach to multi-hop aggregation, by recursively composing BARGs with rate-1. For a tree T of verification keys and messages, consisting of a root node and subtrees $T_1, \ldots, T_k$ the aggregated signature is a BARG proving that for each i, there exists a valid aggregate signature for T_i. Therefore, verification requires the full structure of the tree T. To argue unforgeability, they sample a random leaf in the tree, and program the BARG in each level to extract a proof for the subtree containing this leaf. This requires the tree structure to be statically fixed at the time of CRS generation (and its size to be polynomially bounded).

To go beyond these limitations, we turn to our framework of mergeable trapdoor languages. Our idea is inspired by [BCJP24], which used functional BARGs to construct single-hop monotone-policy aggregate signatures. We define a language $L \in \mathsf{NP}$ where each instance is a key-message pair, and a witness is a valid signature. Our goal is to define an instance merging procedure (that is, a method of aggregating keys and messages) such that given any aggregated key-message pair, we can use the trapdoor to efficiently decide if this aggregated pair is "good", in the sense that it contains a leaf $(\mathsf{vk}^*, \mathsf{m}^*)$ such that vk^* is our challenge verification key, and m^* is a message that was not queried to the signing oracle. This allows us to

use trapdoor extraction to extract such a good pair together with a witness (i.e., a valid signature), which would give us an adversary against the underlying signature scheme.

Towards defining the instance merger IM and trapdoor language D, we first consider the easier problem of selective security, where the message m^* is fixed in advance. The high-level idea is to make the public parameters "programmable" on challenge key-message pair in a way that is indistinguishable to the adversary. We define the public parameters β to contain an FHE encryption of zero and the trapdoor is the FHE secret key. Looking ahead, in the reduction this ciphertext will encrypt the challenge verification key and message $\mathsf{FHE.Enc}(\mathsf{vk}^*, \mathsf{m}^*)$. The instance merger, given a key-message pair $(\mathsf{vk}, \mathsf{m})$, homomorphically computes an encryption of a bit indicating whether or not $(\mathsf{vk}, \mathsf{m}) = (\mathsf{vk}^*, \mathsf{m}^*)$. We merge such aggregated keys and messages by homomorphically evaluating the logical OR of these bits. The language $D \in P$ contains all ciphertexts that decrypt to 1. Therefore, if β encrypts $(\mathsf{vk}^*, \mathsf{m}^*)$, an aggregated keys and messages will belong to D if and only if they contain the pair $(\mathsf{vk}^*, \mathsf{m}^*)$.

Finally, to go from selective to adaptive security for the message, our solution is inspired by the admissible hash paradigm [BB04, Wat05]. The idea is to program the public parameters on a "random" predicate that, with some noticeable probability, is satisfied by the challenge message, but not by any of the other message queried by the adversary. Since the adversary can select its queries adaptively, we crucially rely on the fact that the programmed predicate is hidden from the adversary. In more detail, our predicate P is given by a PRF key k and a number q such that $P(\mathsf{m}) = 1$ iff $\mathsf{PRF}(\mathsf{k}, \mathsf{m}) \equiv 0 \bmod q$. The public parameters β contain an encryption of P and of the challenge verification key. The merger checks homomorphically that $P(\mathsf{m}) = 1$ and $\mathsf{vk} = \mathsf{vk}^*$. We set q in the reduction, depending on the number of signing queries of the adversary such that with noticeable probability over the choice of k, all messages m that were queried to the signing oracle have $P(\mathsf{m}) = 0$ while for the challenge message $P(\mathsf{m}^*) = 1$.

Comparison with [DGKV22]. In contrast to the construction of [DGKV22], in our multi-hop aggregate signature scheme one can verify an aggregated signature without knowing the structure of the entire merge tree, instead verification only requires a short aggregated key-message pair $(\hat{\mathsf{vk}}, \hat{\mathsf{m}})$. This also allows us to consider a stronger notion of unforgeability. Recall that to win the aggregate unforgeability game, the adversary must produce $(\hat{\mathsf{vk}}, \hat{\mathsf{m}})$ together with a valid aggregated signature such that at least one of the pairs aggregated $(\mathsf{vk}^*, \mathsf{m}^*)$ was not given as a query to the signing oracle. Instead of requiring the adversary to produce the entire aggregation tree, we only require the adversary to prove that $(\hat{\mathsf{vk}}, \hat{\mathsf{m}})$ contains $(\mathsf{vk}^*, \mathsf{m}^*)$ by demonstrating an "authentication path" in the aggregation tree. We can satisfy this stronger notion since a local opening of $(\hat{\mathsf{vk}}, \hat{\mathsf{m}})$ to $(\mathsf{vk}^*, \mathsf{m}^*)$ implies that $(\hat{\mathsf{vk}}, \hat{\mathsf{m}}) \notin D$.

Moreover, [DGKV22] consider a security notion where the structure of the aggregation tree is fixed before the public parameters are generated and to win the adversary must follow this aggregation tree. In contrast, our notion of security puts no restrictions on the way the adversary generates the aggregated signature and aggregated key-message pairs.

1.2.5 IVC with universal soundness for trapdoor languages.

Our third application is to incrementally verifiable computation with universal soundness. Our construction closely follows the IVC for P construction in [PP22, DGKV22, Val08]. To prove a statement $(x_0, x_i, i) \in L^*$ these works implicitly rely on mergeable arguments where the size of the merge tree is i and its depth is $\log i$. By relying instead on our mergeable arguments for trapdoor languages, we obtain universal soundness since the running time of our soundness reduction is independent of the tree size i.

For completeness we recall the IVC construction from [PP22, DGKV22, Val08] presenting it based on mergeable arguments for L^*. More specifically, we consider the following instance merger for L^*:

$\mathsf{IM}((x_1, x_2, i), (x_2', x_3, j))$ outputs $(x_1, x_3, i + j)$ if $x_2 = x_2'$, and $\perp$ otherwise. The proof π_i for the statement $(x_0, x_i, i) \in L^*$ consists of at most $\log i$ mergeable argument proofs $\{\bar{\pi}_j\}_{j \in S}$ where S is the set of indexes such that $i = \sum_{j \in S} 2^j$. If $S = \{j_1, \ldots, j_\ell\}$ then each $\bar{\pi}_{j_k}$ proves a statement of the form $(x_{i_{k-1}}, x_{i_k}, 2^j) \in L^*$ where $i_k = 2^{j_1} + \cdots + 2^{j_k}$. Indeed, since $i_0 = 0$ and $i_\ell = i$, the proofs $\bar{\pi}_{j_1}, \ldots, \bar{\pi}_{j_\ell}$ together prove the statement $(x_0, x_i, i) \in L^*$.

Given a proof π_i for $(x_0, x_i, i) \in L^*$ and any z_i, we compute the next proof π_{i+1} as follows. First, we compute the next configuration $x_{i+1} = S(x_i, z_i)$ and we think of $\bar{\pi} = z_i$ as a proof for the statement $(x_i, x_{i+1}, 1) \in L^*$. We add $\bar{\pi}$ to π_i and then, as long as π_i contains proofs for two statements of the form $(x, x', 2^j), (x', x'', 2^j)$ we use the proof merger to merge the two proofs into a proof for the statement $(x, x'', 2^{j+1})$. This process terminates after at most $\log i$ merges and we obtain the next proof π_{i+1}. Indeed, it follows that each $\bar{\pi}_j$ in π_i is computed following a merge tree of size i and depth $\log i$.

2 Preliminaries

2.1 Functional Batch Arguments

We recall the definition of functional batch arguments with strong somewhere extraction, which were introduced in [PP26]. The definition is taken verbatim from [PP26].

Let L_{U} be the language that contains an instance $(\mathcal{M}, z, T)$ if and only if the machine $\mathcal{M}(z)$ accepts in at most T steps. Let $\mathcal{R}_{\mathsf{B}}$ be the relation that contains an instance $x = (\mathcal{M}, k, T)$ with witness $w = (w_1, \ldots, w_k)$ if and only if for every $i \in [k]$, $(\mathcal{M}, (i, w_i), T) \in L_{\mathsf{U}}$.

Let $F = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of functions, where each $\mathcal{F} \in F_\lambda$ is a Turing machine with running time bounded by a polynomial $s(\lambda, \cdot)$. A functional batch argument (BARG) for F consists of polynomial-time algorithms (Gen, Prove, Verify, Extract) with the following syntax:

Gen: The randomized generation algorithm takes as input the security parameter 1^λ and a machine $\mathcal{F} \in F_\lambda$. It outputs a public key pk and a trapdoor td.

Prove: The deterministic prover algorithm takes as input the public key pk, a tag z, instance $x = (M, k, 1^T)$ and a witness w. It outputs a proof π.

Verify: The deterministic verifier algorithm takes as input the public key pk, a tag z, instance $x = (M, k, T)$, and a proof π. It outputs an acceptance bit.

Extract: The deterministic extraction algorithm takes as input the trapdoor td and a proof π. It outputs a witness $w \in \{0,1\}^*$.

Definition 2.1. *A functional BARG* (Gen, Prove, Verify, Extract) *satisfies the following requirements:*

Completeness. *For every $\lambda \in \mathbb{N}$, machine $\mathcal{F}$, tag $z \in \{0,1\}^*$ and $(x = (\mathcal{M}, k, T), (w_1, \ldots, w_k)) \in \mathcal{R}_{\mathsf{B}}$ such that $|\mathcal{F}|, |z|, |\mathcal{M}|, k, T \leq 2^\lambda$, we have that:*

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, z, (M, k, T), \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{pk}, z, (M, k, 1^T), w) \end{array}\right] = 1 \quad .$$

Key Indistinguishability. *For every $\mathrm{poly}(\lambda)$-size adversary $\mathcal{A}$ and polynomial $p(\lambda)$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and pair of machines $\mathcal{F}_1, \mathcal{F}_2$ such that $|\mathcal{F}_1| = |\mathcal{F}_2| \leq p(\lambda)$:*

$$\Pr\left[\mathcal{A}(\mathsf{pk}) = b \;\middle|\; \begin{array}{l} b \leftarrow \{0,1\} \\ (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \quad .$$

Somewhere Extraction. *For every* $\mathsf{poly}(\lambda)$*-size adversary* $\mathcal{A}$ *and polynomial* $p(\lambda)$ *there exists a negligible function* μ *such that for every* $\lambda \in \mathbb{N}$*, machine* $\mathcal{F}$ *and* $T \in [p(\lambda)]$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{pk}, z, (\mathcal{M}, k), \pi) = 1 & (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ (\mathcal{M}, (\mathcal{F}(z), w), T) \notin L_{\mathsf{U}} & (z, \mathcal{M}, k, \pi) \leftarrow \mathcal{A}(\mathsf{pk}) \\ & w \leftarrow \mathsf{Extract}(\mathsf{td}, \pi) \end{array}\right] \leq \mu(\lambda) \ .$$

We say that the BARG satisfies strong somewhere extraction if the above holds even when $\mathcal{A}$ *is given both keys* pk *and* td*.*

Efficiency. *In the completeness experiment above, the proof length is* $|\pi| = \mathsf{poly}(\lambda, m)$*, where* $m = \max_i |w_i|$*. We say that the BARG has rate* 1 *if* $|\pi| = m + \mathsf{poly}(\lambda)$*.*

Remark 2.1 (seBARG)**.** We additionally consider the standard notion of somewhere-extractable BARGs. Here the machine $\mathcal{F}$ can only be a constant function, $\mathcal{F}(z) = i$ for some $i \in [k]$. We thus replace $\mathcal{F}$ in Gen with an index, and omit the tag z from Prove and Verify. The security guarantees are identical.

Theorem 2.2 ([PP26])**.** *Assuming BARGs and rate-1 (leveled) fully homomorphic encryption with statistical correctness, there exists a rate-1 functional BARG with strong somewhere extraction.*

Both BARGs and rate-1 fully homomorphic encryption can be obtained under the LWE assumption [CJJ22, BDGM19].

2.2 Homomorphic Encryption

In this section we define homomorphic encryption.

Definition 2.3 (Encryption Scheme)**.** *A public key encryption scheme consists of three* PPT *algorithms (*Setup, Enc, Dec*) satisfying:*

- **Syntax:**
  - $\mathsf{Setup}(1^\lambda)$ *is a probabilistic algorithm that takes as input* 1^λ *and outputs the keys* $\mathsf{sk}, \mathsf{pk} \in \{0,1\}^*$*.*
  - $\mathsf{Enc}_{\mathsf{pk}}(m)$ *is a probabilistic algorithm that takes as input a message* $m \in \{0,1\}^*$ *and a key* $\mathsf{pk} \in \{0,1\}^*$ *and outputs a ciphertext* $ct \in \{0,1\}^*$*.*
  - $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$ *is a deterministic algorithm that takes as input a ciphertext* $\mathsf{ct} \in \{0,1\}^*$ *and a key* $\mathsf{sk} \in \{0,1\}^*$ *and outputs a message* $m \in \{0,1\}^*$*.*
- **Correctness:** *For any* $\lambda \in \mathbb{N}$ *and* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$*, message* $m \in \{0,1\}^*$*,*

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m))) = m$$

- **Semantic Security.** *For any polynomial-size adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{l|l} b = \mathcal{A}_1(\mathsf{st}, \mathsf{ct}) & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ & (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{pk}) \\ & b \leftarrow \{0,1\} \\ & \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

We assume without loss of generality that pk encodes λ, and that an encryption of a single bit $b \in \{0,1\}$ is of length λ.

Definition 2.4 (Fully Homomorphic Encryption)**.** *A fully homomorphic encryption scheme is an encryption scheme with an additional* PPT *algorithm* Eval*, satisfying the following additional properties:*

- **Syntax:** $\mathsf{Eval}_{\mathsf{pk}}(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ *is a deterministic algorithm that takes as input a public key* pk*, a circuit* C *and ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ *and outputs an evaluated ciphertext* ct*.*
- **Evaluation correctness:** *For any* $\lambda \in \mathbb{N}$*, any* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$*, any ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ *such that* $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_i) = m_i$*, and any circuit* C*, if we set* $\mathsf{ct}^* = \mathsf{Eval}_{\mathsf{pk}}(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ *then* $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}^*) = C(m_1, \ldots, m_\ell)$*.*
- **Compactness:** *There exists a fixed polynomial* poly*, such that for any* $\lambda \in \mathbb{N}$*, any* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$*, any messages* $m_1, \ldots, m_\ell \in \{0,1\}^*$*, any ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ *such that* $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_i)$ *and any circuit* C*,*

$$|\mathsf{Eval}_{\mathsf{pk}}(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)| \leq \mathsf{poly}(|C(m_1, \ldots, m_\ell)|, \lambda) \ .$$

Given a fully homomorphic encryption scheme, we can define multi-hop evaluation. Given a DAG D *where each node in the DAG is labeled by a circuit* C *and each input wire is labeled by a ciphertext* ct*, we can compute the multi-hop evaluated ciphertext* ct^**, using* Eval *to compute each node in the DAG. We additionally define a corresponding compactness property.*

- **Multi-hop Compactness:** *There exists a fixed polynomial* poly*, such that for any* $\lambda \in \mathbb{N}$*, any* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$*, any messages* $m_1, \ldots, m_\ell \in \{0,1\}^*$*, any ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ *such that* $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_i)$ *and any DAG* D *as above, denoting the multi-hop evaluated ciphertext by* ct^* *and evaluated message by* m^**, it holds that*

$$|\mathsf{ct}^*| \leq \mathsf{poly}(|m^*|, \lambda) \ .$$

The scheme is **Multi-hop Semi-Compact** *if*

$$|\mathsf{ct}^*| \leq \mathsf{poly}(|m^*|, \lambda, \mathsf{Depth}(D)) \ ,$$

where the depth of the DAG does not account for the depth of the circuit in each individual node.

Next we define the Chosen-Ciphertext Attack 1 security model.

Definition 2.5 (Chosen-Ciphertext Attack 1)**.** *Let* (Setup, Enc, Dec) *be an encryption scheme. We say the scheme is CCA1 secure if for any oracle-aided polynomial-size adversary* $(\mathcal{A}, \mathcal{A}') = \left\{(\mathcal{A}_\lambda, \mathcal{A}'_\lambda)\right\}_{\lambda \in \mathbb{N}}$*, there exists a negligible function* negl*, such that for all* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{lcl} & & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (m_0, m_1, z) \leftarrow \mathcal{A}^{\mathsf{Dec}_{\mathsf{sk}}(\cdot)}(\mathsf{pk}) \\ b' = b & : & |m_0| = |m_1| \\ & & b \leftarrow \{0,1\} \\ & & b' \leftarrow \mathcal{A}'(\mathsf{pk}, z, \mathsf{Enc}_{\mathsf{pk}}(m_b)) \end{array}\right] \leq \mathsf{negl}(\lambda) \ ,$$

Where $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ *the a decryption oracle that on input* ct *computes and returns* $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$*.*

2.3 Indistinguishability Obfuscation

We now define Indistinguishability Obfuscation ($i\mathcal{O}$) for circuits. This notion guarantees that the obfuscations of two circuits are computationally indistinguishable, as long as the circuits are of the same size and are functionally equivalent.

Definition 2.6 (Indistinguishability Obfuscation ($i\mathcal{O}$) For Circuits)**.** *A uniform* PPT *algorithm $i\mathcal{O}$ is called indistinguishability obfuscator for polynomial-sized circuits if the following holds:*

- **Correctness:** *For every $\lambda \in \mathbb{N}$, circuit C, and circuit C' in the support of $i\mathcal{O}(1^\lambda, C)$, the circuits C and C' are functionally equivalent.*
- (T, ε)**-Security:** *For every two circuit ensembles $\{C_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\{C_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ of polynomial-sized circuits where for every $\lambda \in \mathbb{N}$, $C_{1,\lambda}$ and $C_{1,\lambda}$ are of the same size and are functionally equivalent, and for every adversary $\mathcal{A}$ of size at most T for large enough $\lambda \in \mathbb{N}$, it holds that for large enough $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathcal{A}(i\mathcal{O}(1^\lambda, C_{0,\lambda}) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(1^\lambda, C_{1,\lambda}) = 1]\right| \leq \varepsilon(\lambda) \ .$$

We say an $i\mathcal{O}$ scheme is subexponentially *secure if it is $(2^{\lambda^\alpha}, 2^{-\lambda^\alpha})$-secure for some constant $\alpha > 0$.*

2.4 Pseudorandom Generator

We now define pseudorandom generator.

Definition 2.7 (PRG)**.** *Let $m(\cdot)$ be a polynomial. A (T, ε)-pseudorandom generator (PRG) is an efficient mapping $G : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ with the following security: For every adversary $\mathcal{A}$ of size at most T for large enough $\lambda \in \mathbb{N}$, it holds that for any large enough $\lambda \in \mathbb{N}$,*

$$\left|\Pr_{s\leftarrow\{0,1\}^\lambda}[\mathcal{A}(G(s)) = 1] - \Pr_{r\leftarrow\{0,1\}^{m(\lambda)}}[\mathcal{A}(r) = 1]\right| \leq \varepsilon(\lambda) \ .$$

We say G is subexponentially *secure if it is $(2^{\lambda^\alpha}, 2^{-\lambda^\alpha})$-secure for some constant $\alpha > 0$.*

We say the PRG is injective if it is injective as a mapping.

2.5 Puncturable Pseudorandom Function

We now define puncturable pseudorandom functions [SW14].

Definition 2.8 (PPRF)**.** *Let $n(\cdot), m(\cdot)$ be some polynomials. A puncturable pseudorandom function (PPRF) consists of efficient algorithms* PPRF = (Punc, Eval) *where:*

- **Syntax:**
  - $\mathsf{Punc}(s, x)$ *is a deterministic algorithm that punctures a seed s at a point $x \in \{0,1\}^{n(\lambda)}$ and outputs $s\{x\}$. We call $s\{x\}$ the punctured seed at the point x.*
  - $\mathsf{Eval}(s, x)$ *is a deterministic algorithm that takes as input a seed s and query input $x \in \{0,1\}^{n(\lambda)}$ and outputs $y \in \{0,1\}^{m(n)}$.*

- **Correctness:** *For every $\lambda \in \mathbb{N}$ and two distinct inputs $x, x' \in \{0,1\}^{n(\lambda)}$ it holds that*

$$\mathsf{Eval}(s, x') = \mathsf{Eval}(s\{x\}, x') \ ,$$

where $s \leftarrow \{0,1\}^\lambda$ and $s\{x\} \leftarrow \mathsf{Punc}(s, x)$.

- (T, ε)**-Pseudorandomness at the punctured point:** *For every adversary $\mathcal{A}$ of size at most T for large enough $\lambda \in \mathbb{N}$, it holds that for any large enough $\lambda \in \mathbb{N}$ and any $x \in \{0,1\}^{n(\lambda)}$,*

$$\left|\Pr\left[\ \mathcal{A}(x, s\{x\}, \mathsf{Eval}(s, x)) = 1 \quad : \quad s \leftarrow \{0,1\}^\lambda \ \right]\right.$$
$$\left. - \Pr\left[\ \mathcal{A}(x, s\{x\}, r) = 1 \quad : \quad \begin{matrix} s \leftarrow \{0,1\}^\lambda \\ r \leftarrow \{0,1\}^{m(\lambda)} \end{matrix} \ \right]\right| \leq \varepsilon(\lambda) \ ,$$

We say a PPRF *scheme is* subexponentially *secure if it is $(2^{\lambda^\alpha}, 2^{-\lambda^\alpha})$-secure for some constant $\alpha > 0$.*

Next, we can build a PPRF from OWF, for example by the GGM construction [GGM86].

Theorem 2.9 (OWF imply PPRF, [GGM86,SW14])**.** *Assuming subexponential OWF, there exists subexponential PPRF scheme.*

2.6 Non-interactive Zero Knowledge

In this section we define non-interactive zero knowledge. Let $\mathcal{R}$ be an NP relation, and let L be the associated language $L = \{x \in \{0,1\}^* \mid \exists w \in \{0,1\}^* \ s.t. \ (x, w) \in \mathcal{R}\}$. A non-interactive zero knowledge for $\mathcal{R}$ consists of algorithms (Setup, Prove, Verify, Sim) with the following syntax:

Setup**:** The setup algorithm takes as input the security parameter 1^λ and instance size 1^n. It outputs a common reference string crs.

Prove**:** The prover algorithm takes as the common reference string crs, an instance x and a witness w. It outputs a proof π.

Verify**:** The deterministic verifier algorithm takes as input a common reference string crs, an instance x, and a proof π. It outputs an acceptance bit.

Sim**:** A simulation algorithm is a pair of algorithms $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$. The algorithm Sim_1 takes as input the security parameter 1^λ and instance size 1^n. It outputs a simulated common reference string crs and a trapdoor τ. The algorithm Sim_2 takes as input a simulated common reference string crs, a trapdoor τ, and an instance x. It outputs a simulated proof π.

Definition 2.10. *A non-interactive zero knowledge* (Setup, Prove, Verify, Sim) *satisfies the following requirements:*

Completeness. *For every $\lambda \in \mathbb{N}$ and $(x, w) \in \mathcal{R}$ where $|x| = n$ it holds that:*

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{matrix} \ \right] = 1 \ .$$

Adaptive Soundness. *For every* $\mathsf{poly}(\lambda)$*-size adversary* $\mathcal{A}$ *there exists a negligible function* μ *such that for every* $\lambda \in \mathbb{N}$ *it holds that:*

$$\Pr\left[\begin{array}{lll} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ & & (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & & x \notin L \end{array}\right] \leq \mu(\lambda) \ .$$

We say the scheme has statistical soundness if there exists a negligible function μ *such that the above hold even for unbounded adversary* $\mathcal{A}$*.*

Adaptive Zero Knowledge. *For every* $\mathsf{poly}(\lambda)$*-size adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* μ *such that for every* $\lambda \in \mathbb{N}$ *and* $n \in \mathbb{N}$ *it holds that:*

$$\left|\Pr\left[\begin{array}{lll} \mathcal{A}_2(\mathsf{crs}, x, \pi, z) = 1 & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ & & (x, w, z) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ & & \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right]\right.$$
$$\left. - \Pr\left[\begin{array}{lll} \mathcal{A}_2(\mathsf{crs}, x, \pi, z) = 1 & : & (\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^n) \\ & & (x, w, z) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ & & \pi \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \tau, x) \end{array}\right]\right| \leq \mu(\lambda) \ .$$

In the above experiment, $\mathcal{A}_1$ *is required to output* $(x, w) \in \mathcal{R}$*.*

We can build NIZK from LWE [PS19]:

Theorem 2.11 ([PS19])**.** *Assuming the hardness of* LWE *with suitable polynomial factors, for any NP language there exists a statistically sound* NIZK*.*

2.7 Digital Signatures

In this section we define (standard) digital signatures.

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

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

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

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

Definition 2.12. *A digital signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is required to satisfy the following properties:*

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

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

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

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

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

Remark 2.2 (Message space)**.** We assume that the message space is $\{0,1\}^\lambda$. This is without loss of generality, since we can sign arbitrary messages by first applying a hash function (that satisfies targeted collision resistance), then signing the hashed message [NY89].

3 Mergeable Arguments

Let $L = \{L_\beta\}_{\beta \in \mathrm{B}}$ be language family where $\mathrm{B} = \{\mathrm{B}_\lambda\}_{\lambda \in \mathbb{N}}$ is an ensemble of polynomial size indexes. We use the following terminology.

- $L \in P$ if there exists a polynomial-time deterministic algorithm D_L such that for every $\beta \in \mathrm{B}_\lambda$, $x \in L_\beta$ if and only if $\mathsf{D}_L(\beta, x) = 1$.
- $L \in NP$ if there exists a deterministic algorithm V_L such that for every $\beta \in \mathrm{B}_\lambda$, $x \in L_\beta$ if and only if there exists w such that $\mathsf{V}_L(\beta, x, w) = 1$. Moreover, V_L runs in time $\mathrm{poly}(\lambda, |x|)$.

Let L be a language family. An *instance merger* IM for L is a deterministic algorithm that takes as input an index $\beta \in \mathrm{B}$ and a sequence of instances $x_1, \ldots, x_k$. It outputs an instance x or $\perp$.

Definition 3.1. *A polynomial-time algorithm* IM *is an instance merger for L if there exists an efficiently computable weight function $\mathrm{W} : \{0,1\}^* \to \mathbb{N}$ such that if $\mathsf{IM}(\beta, (x_1, \ldots, x_k))$ outputs $x \neq \perp$ then $\mathrm{W}(x) \geq \max_i \mathrm{W}(x_i) + 1$.*

Let $L = \{L_\beta\}_{\beta \in \mathrm{B}}$ and $L^* = \{L^*_\tau\}_{\tau \in \mathrm{T}}$ be language families. A (possibly inefficient) mapping τ from L to L^* is a function ensemble $\tau = \{\tau_\lambda : \mathrm{B}_\lambda \to \mathrm{T}_\lambda\}_{\lambda \in \mathbb{N}}$.

Definition 3.2. *Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families with a mapping τ from L to L^*. Let* IM *be an instance merger for L. We say that:*

- *τ is sound for L if for every $\beta \in \mathrm{B}$, $L_\beta \subseteq L^*_{\tau(\beta)}$.*
- *τ is sound for* IM *if for every $\beta \in \mathrm{B}$ and $x_1, \ldots, x_k \in L^*_{\tau(\beta)}$ it holds that $\mathsf{IM}(\beta, (x_1, \ldots, x_k)) \in L^*_{\tau(\beta)} \cup \{\perp\}$.*

Remark 3.1 (Instance merger with auxiliary input)**.** The instance merger may additionally receive an auxiliary input z, used to specify the type of merge operation to be performed. In this case, τ is sound for IM if for every $\beta \in \mathrm{B}$, auxiliary input z and $x_1, \ldots, x_k \in L^*_{\tau(\beta)}$ it holds that $\mathsf{IM}(\beta, z, (x_1, \ldots, x_k)) \in L^*_{\tau(\beta)} \cup \{\perp\}$.

We can handle such instance mergers without loss of generality by pushing the auxiliary input into the first instance, and extending the language to include such instances. In more detail, given a language L, we consider the extended language $\widehat{L} = \{(z, x) |\ s.t.\, x \in L\}$. The auxiliary input z is parsed as self-delimiting. Then, given instances $\hat{x}_1 = (z_1, x_1), ..., \hat{x}_k = (z_k, x_k)$, the instance merger computes $\mathsf{IM}(\beta, \hat{x}_1, ..., \hat{x}_k) = \mathsf{IM}(\beta, z_1, x_1, ..., x_k)$.

Let $L \in NP$ be a language family. A non-interactive mergeable argument for L consists of algorithms (Gen, Prove, Verify, PM) with the following syntax:

Key generation: The probabilistic key generation algorithm Gen takes as input a security parameter 1^λ and an instance length bound $n \in \mathbb{N}$. It outputs a public key pk.

Prover: The deterministic prover algorithm Prove takes as input the public parameters β, the public key pk, an instance $x \in L_\beta$, and a witness $w \in \{0,1\}^*$. It outputs a proof π.

Verifier: The deterministic verifier algorithm Verify takes as input the public parameters β, the public key pk, an instance x and a proof π. It outputs a bit indicating if it accepts or rejects.

Proof merger: The deterministic proof merger algorithm PM takes as input the public parameters β, the public key pk and a sequence of pairs $(x_1, \pi_1), \ldots, (x_k, \pi_k)$. It outputs a proof π.

Definition 3.3. *Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families and let IM be an instance merger for L. A non-interactive mergeable argument for (L, L^*, IM) satisfies the following requirements:*

Completeness: *For every $\beta \in \mathrm{B}_\lambda$, $n \leq 2^\lambda$, instance x and witness w such that $|x| \leq n$ and $\mathrm{V}_L(\beta, x, w) = 1$:*

$$\Pr\left[\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \pi \leftarrow \mathsf{Prove}(\beta, \mathsf{pk}, x, w) \end{array}\right] = 1 \quad .$$

Efficiency: *In the completeness experiment above the key generation, prover and verifier algorithms run in time* $\mathsf{poly}(\lambda, n)$.

Merge Completeness: *For every $\beta \in \mathrm{B}_\lambda$, $n, k \leq 2^\lambda$, $\mathsf{pk} \in \mathsf{Gen}(1^\lambda, n)$ and $(x_1, \pi_1), \ldots, (x_k, \pi_k)$ such that $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi_i) = 1$ for every $i \in [k]$, let*

$$\begin{aligned} x &\leftarrow \mathsf{IM}(\beta, (x_1, \ldots, x_k)) \\ \pi &\leftarrow \mathsf{PM}(\beta, \mathsf{pk}, (x_1, \pi_1), \ldots, (x_k, \pi_k)) \quad . \end{aligned}$$

If $x \neq \perp$ then $\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1$.

Merge Efficiency: *In the merge completeness experiment above, PM runs in time polynomial in the security parameter λ and its input size, and*

$$|\pi| \leq |(x_1, \ldots, x_k)| \cdot \mathsf{poly}(\lambda) + \max_i |\pi_i| \quad .$$

We say the argument is depth-compact if:

$$|\pi| \leq \max_i |\pi_i| \quad .$$

Remark 3.2 (Merge efficiency for instance mergers with auxiliary input)**.** Note that if the instance merger accepts auxiliary input z (see Remark 3.1), the proof size grows with the size of the auxiliary input, as the auxiliary input is pushed to the instance x to create an extended instance $\hat{x} = (z, x)$.

Soundness: *For every polynomials* $\bar{W}(\lambda), n(\lambda)$*, and poly-size adversary* $\mathcal{A}$ *there exists a negligible function* μ *such that for every mapping* τ *be a mapping from* L *to* L^* *that is sound for* L *and for* IM*, and every* $\lambda \in \mathbb{N}$ *and* $\beta \in B_\lambda$*:*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1 & \mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda, n(\lambda)) \\ W(x) \leq \bar{W}(\lambda) & (x, \pi) \leftarrow \mathcal{A}(\mathsf{pk}) \\ x \notin L^*_{\tau(\beta)} & \end{array}\right] \leq \mu(\lambda) \ .$$

We say the scheme has universal soundness if above requirement holds for $\bar{W}(\lambda) = 2^\lambda$*.*

For convenience, if the running time of Gen is $\mathrm{poly}(\lambda, \log n)$ we set $n = 2^\lambda$ and remove the input n to Gen.

In Section 4.1 and Section 4.3, we prove the following:

Theorem 3.4. *Let* $L \in \mathsf{NP}$ *and* $L^* \in P$ *be language families and let* IM *be an instance merger for* L*. Assuming there exist rate-1 functional BARGs with strong somewhere-extraction, there exists a non-interactive mergeable argument for* (L, L^*, IM) *with universal soundness.*

In Section 5 we prove the following:

Theorem 3.5. *Let* $L \in \mathsf{NP}$ *and* $L^* \in P$ *be language families and let* IM *be an instance merger for* L*. Assuming there exist:*

- *A subexponentially secure indistinguishability obfuscator.*
- *A subexponentially secure one-way function.*
- *An injective PRG.*

there exists a non-interactive depth-compact mergeable argument for (L, L^*, IM)*. If the PRG is subexponentially secure, there exists a scheme with universal soundness.*

3.1 Trapdoor Extraction

We extend the notion of mergeable argument to support mapping that are not sound for L. For such mapping we cannot hope to get soundness. Instead, we define a notion of *trapdoor extraction*.

Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families and let IM be an instance merger for L. A non-interactive mergeable argument $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{PM})$ for (L, L^*, IM) with *trapdoor extraction* also consists of algorithms $(\mathsf{TGen}, \mathsf{Extract})$ with the following syntax:

Trapdoor key generation: The probabilistic trapdoor key generation algorithm TGen takes as input a security parameter 1^λ, an instance length bound $n \in \mathbb{N}$ and a trapdoor τ. It outputs a public key pk and an extraction trapdoor td.

Extraction: The deterministic extraction algorithm Extract takes as input the extraction trapdoor td, an instance x and a proof π. It outputs an instance x' and a witness w'.

Definition 3.6. *Let* $L \in \mathsf{NP}$ *and* $L^* \in P$ *be language families and let* IM *be an instance merger for* L*. A non-interactive mergeable argument for* (L, L^*, IM) *with trapdoor extraction satisfies the requirements in Definition 3.3 and, in addition, it satisfies the following requirements:*

Trapdoor hiding: *For every polynomial $n(\lambda)$, and poly-size adversary $\mathcal{A}$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $\tau \in \mathrm{T}_\lambda$:*

$$\Pr\left[\mathcal{A}(\mathsf{pk}_b) = b \;\middle|\; \begin{array}{l} b \leftarrow \{0,1\}, \\ \mathsf{pk}_0 \leftarrow \mathsf{Gen}(1^\lambda, n(\lambda)) \\ (\mathsf{pk}_1, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, n(\lambda), \tau) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \ .$$

Trapdoor extraction: *For every polynomials $\bar{\mathsf{W}}(\lambda), n(\lambda)$, and poly-size adversary $\mathcal{A}$ there exists a negligible function μ such that for every mapping τ from L to L^* that is sound for IM, and every $\lambda \in \mathbb{N}$ and $\beta \in \mathrm{B}_\lambda$:*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1 \\ \mathsf{W}(x) \leq \bar{\mathsf{W}}(\lambda) \\ x \notin L^*_{\tau(\beta)} \end{array} \rightarrow \begin{array}{l} x' \notin L^*_{\tau(\beta)} \\ \mathsf{V}_L(\beta, x', w') = 1 \end{array} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, n(\lambda), \tau(\beta)) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{pk}) \\ (x', w') \leftarrow \mathsf{Extract}(\mathsf{td}, x, \pi) \end{array}\right] \geq 1 - \mu(\lambda) \ .$$

Furthermore, in the above experiment, Extract *runs in time polynomial in its input length and in $\mathsf{W}(x)$.*

We say the scheme has universal trapdoor extraction if the above requirement holds for $\bar{\mathsf{W}}(\lambda) = 2^\lambda$, and Extract *runs in time polynomial in its input length.*

Remark 3.3 (Trapdoor extraction and soundness)**.** We note that the two requirements above imply the standard soundness requirement for mappings τ that are sound for L. Indeed, if τ is sound for L then $L_\beta \subseteq L^*_{\tau(\beta)}$, which implies that the consequent in the trapdoor extraction experiment is a contradiction, and thus the antecedent only happens with negligible probability. Using trapdoor hiding, and since $x \notin L^*_{\tau(\beta)}$ is efficiently decidable, the same holds if $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda, n(\lambda))$, thus implying soundness.

In Section 4.1 and Section 4.3, we prove the following:

Theorem 3.7. *Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families and let IM be an instance merger for L. Assuming there exist rate-1 functional BARGs with strong somewhere-extraction, there exists a non-interactive mergeable argument for (L, L^*, IM) with universal trapdoor extraction.*

4 Mergeable Arguments from BARGs

4.1 Construction

In this section we describe our first construction of mergeable arguments based on a functional batch argument for NP. Our construction also satisfies trapdoor extraction.

Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families. Let V_L be the verifier of L, and let D_{L^*} be the decider of L^*. Let IM be an instance merger for L. Let $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}_{\mathsf{BARG}}, \mathsf{Prove}_{\mathsf{BARG}}, \mathsf{Verify}_{\mathsf{BARG}}, \mathsf{Extract}_{\mathsf{BARG}})$ be a rate-1 functional BARG. The construction of $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{PM}, \mathsf{TGen}, \mathsf{Extract})$ is as follows:

Gen : On input security parameter 1^λ, the generation algorithm is as follows:

- Let $\mathcal{F}_\perp$ be a machine that always outputs $\perp$ padded to size $p(\lambda)$ where p is a polynomial specified in the analysis.
- Sample BARG keys $(\mathsf{pk}, \gamma) \leftarrow \mathsf{Gen}_{\mathsf{BARG}}(1^\lambda, \mathcal{F}_\perp)$.
- Output pk.

Prove: On input the public parameters β, public key pk, instance x, and a witness w, the prover algorithm is as follows:

- Output $(\emptyset, w)$.

Verify: On input the public parameters β, public key pk, instance x, and a proof π, the verifier algorithm is as follows:

- Parse π as (z, π').
- If $z = \emptyset$ then output $\mathsf{V}_L(\beta, x, \pi')$.
- Parse z as $(x_1, ..., x_k)$.
- If $x \neq \mathsf{IM}(\beta, (x_1, ..., x_k))$ then output 0.
- Let $\mathcal{M}[\mathsf{pk}, \beta, z]$ be a machine that given (i, π^*) outputs $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi^*)$, and let T be a bound on its running time.
- Output $\mathsf{Verify}_{\mathsf{BARG}}(\mathsf{pk}, z, (\mathcal{M}[pk, \beta, z], k, T), \pi')$.

PM: On input the public parameters β, public key pk, and tuples $(x_1, \pi_1), ..., (x_k, \pi_k)$, the proof merger algorithm is as follows:

- Set $z = (x_1, ..., x_k)$.
- Let $\mathcal{M}[\mathsf{pk}, \beta, z]$ be a machine that given (i, π^*) outputs $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi^*)$, and let T be a bound on its running time.
- Output $(z, \mathsf{Prove}_{\mathsf{BARG}}(\mathsf{pk}, z, (\mathcal{M}[pk, \beta, z], k, 1^T), (\pi_1, ..., \pi_k)))$.

TGen: On input security parameter 1^λ and trapdoor τ, the trapdoor key generation algorithm is as follows:

- Let $\mathcal{F}[\tau]$ be the machine that on input $(x_1, ..., x_k)$, outputs:
  - 1, if for all $0 \leq i \leq k$, $\mathsf{D}_{L^*}(\tau, x_i) = 1$.
  - i, where i is the first index where $\mathsf{D}_{L^*}(\tau, x_i) = 0$.

  The machine $\mathcal{F}[\tau]$ is padded to size $p(\lambda)$, where p is a polynomial specified in the analysis.
- Sample $(\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\mathsf{BARG}}(1^\lambda, \mathcal{F}[\tau])$.
- Output $(\mathsf{pk}, (\mathcal{F}[\tau], \mathsf{td}))$.

Extract: On input extraction trapdoor td, and instance x and proof π, the extraction algorithms is as follows:

- Parse td as $(\mathcal{F}, \mathsf{td}')$.
- Set $j = 0$.
- Set $x'_j = x$.
- Set $\pi_j = \pi$.
- Parse π_j as (z_0, π'_0).
- While $z_j \neq \emptyset$:
  - Set $j = j + 1$.
  - Parse z_{j-1} as $(x_1, ..., x_k)$.
  - If $\mathsf{IM}(\beta, (x_1, ..., x_k)) \neq x'_{j-1}$ then abort and output $\perp$.

- Compute $i_j \leftarrow \mathcal{F}(z_{j-1})$.
- Set $x'_j = x_{i_j}$.
- Extract $\pi_j \leftarrow \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}', \pi'_{j-1})$
- Parse π_j as (z_j, π'_j).

• Output (x'_j, π'_j).

Theorem 4.1. *Assuming* Π_{BARG} *is a rate-*1 *functional* BARG *with strong somewhere-extraction, then* Π *is a mergeable argument with trapdoor extraction.*

We prove the theorem via the following claims:

Claim 4.2 (Completeness and Efficiency)**.** *Assuming* Π_{BARG} *is a rate-*1 *functional* BARG *with strong somewhere-extraction, then* Π *satisfies completeness, merge completeness and merge efficiency.*

Claim 4.3. *Assuming* Π_{BARG} *is a rate-*1 *functional* BARG *with strong somewhere-extraction, then* Π *satisfies trapdoor hiding and trapdoor extraction.*

Proof of Claim 4.2.
Completeness and Merge Completeness. Completeness follows directly from the completeness of the verifier V_L. Merge completeness also follows directly by construction and the correctness of the underlying BARG scheme. In particular, given a public parameter β, and merged instance x and merged proof $\pi = (z, \pi')$, note that $x = \mathsf{IM}(\beta, z)$.

Merge Efficiency. Let pk, β, and $(x_1, \pi_1), ..., (x_k, \pi_k)$ be the inputs to PM, and let $\pi = (z, \pi')$ be the output of PM, where $z = (x_1, ..., x_k)$ and

$$\pi' = \mathsf{Prove}_{\mathsf{BARG}}(\mathsf{pk}, z, (\mathcal{M}[pk, \beta, z], k, 1^T), (\pi_1, ..., \pi_k)) \ .$$

Since the BARG is rate-1, there exists a polynomial p such that $|\pi'| \leq \max_i |\pi_i| + p(\lambda)$. It therefore follows that

$$|\pi| \leq |(x_1, ..., x_k)| + \max_i |\pi_i| + p(\lambda) \ ,$$

which implies merge efficiency. □

Proof of Claim 4.3.
Trapdoor hiding follows directly from the key indistinguishability of the BARG scheme. We therefore proceed to prove trapdoor extraction. Let τ be a mapping from L to L^* that is sound for IM. Let $t(\lambda)$ be a bound on $|\tau(\beta_\lambda)|$, for any $\beta_\lambda \in \mathrm{B}_\lambda$. Let $\bar{\mathrm{W}}(\lambda)$ be a polynomial, and let $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ be poly-size adversary. Let $\beta := \beta_\lambda \in \mathrm{B}_\lambda$.

Let (x, π) be the output of $\mathcal{A}$. Now, if $\mathrm{W}(x) > \bar{\mathrm{W}}(\lambda)$ the condition of the trapdoor extraction is satisfied trivially. Let us focus on the case that $\mathrm{W}(x) \leq \bar{\mathrm{W}}(\lambda)$. Since for any $x_1, ..., x_k$ it holds that $\mathrm{W}(\mathsf{IM}(\beta, (x_1, ..., x_k))) \geq \max_i \mathrm{W}(x_i) + 1$, it follows that the extractor halts after at most $\mathrm{W}(x)$ iterations. Indeed, at each iteration either $x'_j = \mathsf{IM}(\beta, z_j)$ or the extractor aborts. Let $0 \leq j^* \leq \mathrm{W}(x)$ be the number of iterations performed by the extractor until it halts. For $j \in \mathbb{N}$, denote by A_j the event that the extractor did not halt in the first j iterations, and at the beginning of the $j+1$ iteration of the extraction algorithm, the following holds:

$$x'_j \notin L^*_{\tau(\beta)} \ ,$$
$$\mathsf{Verify}(\beta, \mathsf{pk}, x_j, \pi_j) = 1 \ .$$

If A_0 doesn't hold then the condition of the trapdoor extraction is satisfied trivially. Let us focus on the case it holds. Next, note that if $j^* = 0$, then by construction $\pi = (\emptyset, \pi')$, where $\mathsf{V}_L(\beta, x, \pi') = 1$, and again, the condition of the trapdoor extraction is satisfied. We can therefore focus on the case where $j^* \geq 1$. In particular, since when $z_j = \emptyset$ the Verify returns as $\mathsf{V}_L(\beta, x, \pi')$, it is sufficient to show that there exists a negligible function μ such that

$$\Pr\left[A_0 \wedge \bar{A}_{j^*}\right] \leq \mu(\lambda) \ .$$

To that end, we will show that

$$\Pr\left[A_0 \bigwedge (\bigvee_{j=1}^{j^*} \bar{A}_j)\right] \leq \mu(\lambda) \ ,$$

for a negligible function μ. Indeed, note that

$$\Pr\left[A_0 \bigwedge (\bigvee_{j=1}^{j^*} \bar{A}_j)\right] \leq \Pr\left[\bigvee_{j=1}^{\bar{\mathsf{W}}(\lambda)} (\bar{A}_j \wedge A_{j-1} \wedge j^* \geq j)\right] \leq \sum_{j=1}^{\bar{\mathsf{W}}(\lambda)} \Pr\left[\bar{A}_j \wedge A_{j-1} \wedge j^* \geq j\right].$$

By Claim 4.4, there exists a negligible function $\mu_{\bar{\mathsf{W}}}(\cdot)$ such that $\Pr\left[\bar{A}_j \wedge A_{j-1} \wedge j^* \geq j\right] \leq \mu_{\bar{\mathsf{W}}}(\lambda)$. We obtain

$$\sum_{j=1}^{\bar{\mathsf{W}}(\lambda)} \Pr\left[\bar{A}_j \wedge A_{j-1} \wedge j^* \geq j\right] \leq \bar{\mathsf{W}}(\lambda) \cdot \mu_{\bar{\mathsf{W}}}(\lambda) \ ,$$

which is negligible, since $\bar{\mathsf{W}}$ is a polynomial function. □

Claim 4.4. *Assume Π_{BARG} is a rate-1 functional* BARG *with strong somewhere-extraction. Let $\mathcal{A}$ be a poly-size adversary. Let (x, π) be the output of $\mathcal{A}$. Let $j^* \geq 0$ be the number of iterations performed by the extractor until it halts. For $j \in \mathbb{N}$, denote by A_j the event that the extractor did not halt in the first j iterations, and at the beginning of the $j+1$ iteration of the extraction algorithm, the following holds:*

$$x'_j \notin L^*_{\tau(\beta_\lambda)} \ ,$$
$$\mathsf{Verify}(\beta_\lambda, \mathsf{pk}, x_j, \pi_j) = 1 \ .$$

Then for any polynomial $t(\cdot)$, there exists a negligible function $\mu_t(\cdot)$ such that for any polynomial $j(\cdot)$ such that $1 \leq j(\lambda) \leq t(\lambda)$ for any $\lambda \in \mathbb{N}$, and for any mapping $\tau : \mathrm{B}_\lambda \to \mathrm{T}_\lambda$ that is sound for IM, it holds that for any $\lambda \in \mathbb{N}$,

$$\Pr\left[\bar{A}_{j(\lambda)} \wedge A_{j(\lambda)-1} \wedge j^* \geq j(\lambda)\right] \leq \mu_t(\lambda) \ .$$

Proof. In the following let $\beta := \beta_\lambda$. Consider $\pi_{j-1} = (z_{j-1}, \pi'_{j-1})$ and x'_{j-1} at the beginning of the j-th iteration, and $\pi_j = (z_j, \pi'_j)$ and x'_j at the end of the j-th iteration. First, we show that $x'_j \notin L^*_{\tau(\beta_\lambda)}$. Indeed, since we run the j-th iteration, it follows that $z_{j-1} \neq \emptyset$. Parse z_{j-1} as $(x_1, ..., x_k)$. When the event A_{j-1} happens, it must be the case that $x_{j-1} = \mathsf{IM}(\beta, z_{j-1})$, since this condition is checked by Verify. So, since τ is sound for IM by assumption, and since $x'_{j-1} \notin L^*_{\tau(\beta)}$ in the event A_{j-1}, it follows that there exists $1 \leq i \leq k$ such that $x_i \notin L^*_{\tau(\beta)}$. Thus, by the correctness of the decider D_{L^*}, and by the definition of the machine $\mathcal{F}$ constructed by TGen, it follows that $x'_j \notin L^*_{\tau(\beta)}$.

Therefore, in order for the event $\bar{A}_j \wedge A_{j-1} \wedge j^* \geq j$ to happen, it must be the case that $\mathsf{Verify}(\beta, \mathsf{pk}, x_j, \pi_j) = 0$. We bound this probability by a direct reduction to the strong somewhere extractability of the BARG. Consider the following adversary $\mathcal{A}'_j$ to the BARG scheme. $\mathcal{A}'_j$ has j hardwired. Given $(\mathsf{pk}, \mathsf{td})$ generated by $\mathsf{Gen}_{\mathsf{BARG}}(\lambda, \mathcal{F}[\tau(\beta)])$, $\mathcal{A}'_j$ does the following:

- Constructs the machine $\mathcal{F}[\tau(\beta)]$ as in TGen.
- Runs the original adversary and obtain $(x, \pi) \leftarrow \mathcal{A}(\mathsf{pk})$.
- Runs the extraction algorithm Extract for $j-1$ iterations, and obtain $(x'_{j-1}, z_{j-1}, \pi'_{j-1})$.
- If $z_{j-1} = \emptyset$, or the Extract algorithm aborts, $\mathcal{A}$ aborts.
- Parses $z_{j-1} = (x_1, ..., x_k)$.
- Constructs the machine $\mathcal{M} := \mathcal{M}[\mathsf{pk}, \beta, z_{j-1}]$ that given (i, π^*) outputs $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi^*)$, and let T be a bound on its running time.
- Outputs $(z_{j-1}, \mathcal{M}, k, \pi'_{j-1})$.

Note that by construction, if $z_{j-1} \neq \emptyset$ then we extract the next proof: $\pi'_j = \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}, \pi'_{j-1})$. Moreover, the following hold:

- $\mathsf{Verify}(\beta, \mathsf{pk}, x_{j-1}, \pi_{j-1}) = 1$ implies that $\mathsf{Verify}_{\mathsf{BARG}}(\mathsf{pk}, z_{j-1}, (\mathcal{M}, k, T), \pi'_{j-1}) = 1$.
- By the definition of $\mathcal{M}$, $\mathsf{Verify}(\beta, \mathsf{pk}, x_j, \pi_j) = 0$ implies that $\mathcal{M}(i_j, \pi_j) = 0$, hence $(\mathcal{M}, (i_j, \pi_j), T) \notin L_{\mathsf{U}}$, where $i_j = \mathcal{F}[\tau(\beta)](z_{j-1})$.

Therefore, the event $\bar{A}_j \wedge A_{j-1} \wedge j^* \geq j$ implies $\mathcal{A}'_j$ won the somewhere extractability game of the BARG scheme. Observe that the size of $\mathcal{A}'_j$ is polynomial in the size of $\mathcal{A}$. Indeed, the machine $\mathcal{F}[\tau(\beta)]$ it constructs has polynomial size since D_{L^*} is polynomial time algorithm and $\tau(\beta)$ is of polynomial size. Moreover, $\mathcal{A}'_j$ runs the Extract algorithm for at most $j(\lambda) \leq t(\lambda)$ iterations, which is polynomial. Therefore, $\mathcal{A}'_j$ is a polynomial size adversary for any $j \leq t$. Note that $\mathcal{A}'_t$ is the largest adversary. It follows there exists a negligible function $\mu_t(\cdot)$ such $\Pr\left[\bar{A}_{j(\lambda)} \wedge A_{j(\lambda)-1} \wedge j^* \geq j(\lambda)\right] \leq \mu_t(\lambda)$.

Finally, since $\tau(\beta)$ has polynomial size bound $q(\lambda)$, by a standard argument we can set $\mu_t(\lambda)$ to be the one corresponding to the worst mapping τ that is sound for IM. □

4.2 Merge Efficiency for Circuits

In this section we define mergeable arguments with merge efficiency for circuits (Definition 4.5), and give a construction with trapdoor extraction based on LWE. The construction combines the previous construction in Section 4.1, with an additional argument system for computing a circuit in a single hop.

4.2.1 Definition.

We define an additional merge efficiency guarantee, falling between standard efficiency and depth-compact efficiency. Rather than growing with the total depth of the merge circuit (as in standard efficiency), we can compute the circuit in hops comprising more than one gate at a time, and only grow with the number of hops (and the maximum fan-in).

Formally, we extend the notion of the instance merger and proof merger to support computing a circuit of merge gates. A circuit C is given by a DAG, where the input and output nodes are labeled, and each intermediate node represents an application of the instance merger IM. The number of inputs, size and fan-in of the circuit C are denoted by k, s, m respectively.

Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families and let IM be an instance merger for L. A non-interactive mergeable argument $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{PM})$ for (L, L^*, IM) with *merging for circuits* has the following syntactic modifications:

Instance merger for circuits: The deterministic instance merger algorithm IM takes as input the public parameters β, a description of a circuit C, and a sequence of instances $x_1, \ldots, x_k$. It outputs x given by the computation of C on $x_1, \ldots, x_k$, where each gate is an application of the base instance merger IM.

Proof merger for circuits: The deterministic proof merger algorithm PM takes as input the public parameters β, the public key pk, a description of a circuit C, and a sequence of pairs $(x_1, \pi_1), \ldots, (x_k, \pi_k)$. It outputs a proof π.

We note that any mergeable argument can be augmented to support merging for circuits, by iteratively computing the proof merger for each gate. However, in this case (assuming the mergeable argument is not depth-compact), the size of the proof grows with the circuit depth d (in additional to the circuit fan-in m). We require an improved efficiency guarantee, where the size of the proof only grows with the circuit fan-in. We also require an improved soundness guarantee, where soundness holds for polynomially bounded weight, where the weight is only incremented for each circuit computation (rather than for each merge gate).

Definition 4.5. *Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families and let IM be an instance merger for L. A non-interactive mergeable argument for (L, L^*, IM) with merge efficiency for circuits satisfies the requirements in Definition 3.3 and the following additional requirements:*

Merge completeness for circuits: *For every $\beta \in \mathrm{B}_\lambda$, $n, k, s, m \leq 2^\lambda$, $\mathsf{pk} \in \mathsf{Gen}(1^\lambda, n)$, instances and proofs $(x_1, \pi_1), \ldots, (x_k, \pi_k)$ such that $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi_i) = 1$ for every $i \in [k]$, and circuit C of size s and fan-in m, let*

$$x \leftarrow \mathsf{IM}(\beta, \mathsf{C}, (x_1, \ldots, x_k)) \ ,$$
$$\pi \leftarrow \mathsf{PM}(\beta, \mathsf{pk}, \mathsf{C}, (x_1, \pi_1), \ldots, (x_k, \pi_k)) \ .$$

If $x \neq \perp$ then $\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1$.

Merge efficiency for circuits: *In the merge completeness experiment above,*

$$|\pi| \leq m \cdot \max_i |x_i| \cdot \mathsf{poly}(\lambda) + \max_i |\pi_i| \ .$$

Soundness: *Assume there exists a weight function W' such that if $\mathsf{IM}(\beta, \mathsf{C}, (x_1, \ldots, x_k))$ outputs $x \neq \perp$ then $\mathsf{W}'(x) \geq \max_i \mathsf{W}'(x_i) + 1$. Then, soundness holds for this weight function.*

Theorem 4.6. *Assuming leveled FHE and rate-1 functional BARGs with strong somewhere-extraction, there exists a mergeable argument for (L, L^*, IM) with merge efficiency for circuits, and universal trapdoor extraction.*

4.2.2 One-Hop Argument System.

As a building block, we use a helper argument system for proving a circuit of merge gates in one hop. We define the syntax and requirements of this argument system.

Let $L \in \mathsf{NP}, L^* \in P$ be language families with a mapping τ from L to L^*. Let IM be an instance merger, such that τ is sound for IM. The single-hop helper argument system for L^* consists of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract})$ with the following syntax:

Key generation: The probabilistic trapdoor key generation algorithm Gen takes as input a security parameter 1^λ. It outputs a public key pk.

Trapdoor key generation: The probabilistic trapdoor key generation algorithm TGen takes as input a security parameter 1^λ and a trapdoor τ. It outputs a public key pk and an extraction trapdoor td.

Prover: The deterministic prover algorithm Prove takes as input the public parameters β, the public key pk, a circuit C, and instances $x_1, \ldots, x_k \in L^*_\tau$. It outputs a proof π and openings $\rho_1, \ldots, \rho_k$.

Verifier: The deterministic verifier algorithm Verify takes as input the public parameters β, the public key pk, an instance x, a proof π, number of instances k, an index $j \in [k]$, an instance x_j and an opening ρ_j. It outputs a bit indicating if it accepts or rejects.

Extraction: The deterministic extraction algorithm Extract takes as input the extraction trapdoor td, number of instances k and a proof π. It outputs an index $j \in [k]$.

Definition 4.7. *The one-hop helper argument system for (L, L^*, IM) satisfies the following requirements:*

Completeness: *For every $\lambda \in \mathbb{N}$, $\beta \in \mathrm{B}_\lambda$, $n, k, s \le 2^\lambda$, instances $x_1, \ldots, x_k$ such that $|x_i| \le n$, circuit C of size s, and $i \in [k]$,*

$$\Pr\left[\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi, k, i, x_i, \rho_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda) \\ (\pi, \rho_1, \ldots, \rho_k) \leftarrow \mathsf{Prove}(\beta, \mathsf{pk}, \mathsf{C}, x_1, \ldots, x_k) \end{array}\right] = 1 \ .$$

Efficiency: *The proof and openings are of size $m \cdot n \cdot \mathsf{poly}(\lambda)$ where m is the fan-in of the circuit C.*

Trapdoor hiding: *For every polynomial poly-size adversary $\mathcal{A}$ there exists a negligible function μ such that for every $\tau \in \mathrm{T}_\lambda$:*

$$\Pr\left[\mathcal{A}(\mathsf{pk}_b) = b \;\middle|\; \begin{array}{l} b \leftarrow \{0,1\}, \\ \mathsf{pk}_0 \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{pk}_1, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, \tau) \end{array}\right] \le \frac{1}{2} + \mu(\lambda) \ .$$

Strong trapdoor extraction: *Let τ be a mapping from L to L^* that is sound for IM. For any poly-size adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\beta \in \mathrm{B}_\lambda$:*

$$\Pr\left[\begin{array}{l} x \notin L^*_{\tau(\beta)} \ \wedge \ j = j^* \ \wedge \\ \mathsf{Verify}(\beta, \mathsf{pk}, x, \pi, k, j, x_j, \rho_j) = 1 \end{array} \rightarrow \ x_j \notin L^*_{\tau(\beta)} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, \tau(\beta)) \\ (x, \pi, k, j, x_j, \rho_j) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{td}) \\ j^* \leftarrow \mathsf{Extract}(\mathsf{td}, k, \pi) \end{array}\right] \ge 1 - \mathsf{negl}(\lambda) \ .$$

Theorem 4.8. *Assuming* seBARGs *and leveled* FHE*, there exists an argument system satisfying Definition 4.7.*

The proof of this theorem is given in Appendix A.

4.2.3 Construction.

In this section we give the construction of mergeable arguments with merge efficiency for circuits. Our construction uses the following building blocks:

- The one-hop helper argument system (Definition 4.7)

$$(\mathsf{Gen}_{\mathsf{OneHop}}, \mathsf{TGen}_{\mathsf{OneHop}}, \mathsf{Prove}_{\mathsf{OneHop}}, \mathsf{Verify}_{\mathsf{OneHop}}, \mathsf{Extract}_{\mathsf{OneHop}}) \ .$$

- A rate-1 functional BARG (Definition 2.1)

$$(\mathsf{Gen}_{\mathsf{BARG}}, \mathsf{Prove}_{\mathsf{BARG}}, \mathsf{Verify}_{\mathsf{BARG}}, \mathsf{Extract}_{\mathsf{BARG}}) .$$

Next we describe the construction. It is based on the construction in Section 4.1.

Gen: On input security parameter 1^λ, the generation algorithm is as follows:

- Sample a one-hop key $\mathsf{pk}_{\mathsf{OneHop}} \leftarrow \mathsf{Gen}_{\mathsf{OneHop}}(1^\lambda)$.
- Let $\mathcal{F}_\perp$ be a machine that always outputs $\perp$ padded to size $p(\lambda)$ where p is a polynomial specified in the analysis.
- Sample BARG keys $(\mathsf{pk}_{\mathsf{BARG}}, \gamma_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}_{\mathsf{BARG}}(1^\lambda, \mathcal{F}_\perp)$.
- Output $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}})$.

Prove: On input a public key pk, public parameter β, instance x, and a witness w, the prover algorithm is as follows:

- Output $(\emptyset, w)$.

Verify: On input the public parameters β, public key pk, instance x, and a proof π, the verifier algorithm is as follows:

- Parse pk as $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}})$.
- Parse π as (z, π').
- If $z = \emptyset$ then output $\mathsf{V}_L(\beta, x, \pi')$.
- Let $\mathcal{M}[\mathsf{pk}, \beta, z, x]$ be a machine that given (i, x_i, ρ_i, π^*) accepts if all the following checks pass:
  - Parse pk as $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}})$.
  - Check that $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi^*) = 1$.
  - Parse z as $(\pi_{\mathsf{OneHop}}, k)$, and check that $\mathsf{Verify}_{\mathsf{OneHop}}(\beta, \mathsf{pk}_{\mathsf{OneHop}}, x, \pi_{\mathsf{OneHop}}, k, i, x_i, \rho_i) = 1$.

  Let T be a bound on $\mathcal{M}$'s running time.
- Output $\mathsf{Verify}_{\mathsf{BARG}}(\mathsf{pk}_{\mathsf{BARG}}, z, (\mathcal{M}[pk, \beta, z], k, T), \pi')$.

PM: On input the public parameters β, public key pk, circuit C, and tuples $(x_1, \pi_1), ..., (x_k, \pi_k)$, the proof merger algorithm is as follows:

- Parse pk as $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}})$.
- Computed the merged instance $x = \mathsf{IM}(\beta, \mathsf{C}, x_1, \ldots, x_k)$.
- Compute the one-hop proof and openings $(\pi_{\mathsf{OneHop}}, \rho_1, \ldots, \rho_k) \leftarrow \mathsf{Prove}_{\mathsf{OneHop}}(\beta, \mathsf{pk}_{\mathsf{OneHop}}, \mathsf{C}, x_1, \ldots, x_k)$.
- Set $z = (\pi_{\mathsf{OneHop}}, k)$.
- Let $\mathcal{M}[\mathsf{pk}, \beta, z, x]$ be the machine defined by Verify above, and let T be a bound on its running time.
- For $i \in [k]$, set $w_i = (x_i, \rho_i, \pi_i)$.
- Output $(z, \mathsf{Prove}_{\mathsf{BARG}}(\mathsf{pk}_{\mathsf{BARG}}, z, (\mathcal{M}[pk, \beta, z], k, 1^T), (w_1, ..., w_k)))$.

TGen: On input security parameter 1^λ and trapdoor τ, the trapdoor key generation algorithm is as follows:

- Sample one-hop keys $(\mathsf{pk}_{\mathsf{OneHop}}, \mathsf{td}_{\mathsf{OneHop}}) \leftarrow \mathsf{TGen}_{\mathsf{OneHop}}(1^\lambda, \tau)$.
- Let $\mathcal{F}[\mathsf{td}_{\mathsf{OneHop}}]$ be the machine that on input π_{OneHop}, k, outputs $\mathsf{Extract}(\mathsf{td}_{\mathsf{OneHop}}, k, \pi_{\mathsf{OneHop}})$.
- Sample $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}_{\mathsf{BARG}}(1^\lambda, \mathcal{F}[\mathsf{td}_{\mathsf{OneHop}}])$.
- Output $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}}), \mathsf{td} = \mathsf{td}_{\mathsf{BARG}}$.

$\mathsf{Extract}$: On input the extraction trapdoor td, instance x and proof π, the extraction algorithm is as follows:

- Parse td as $\mathsf{td}_{\mathsf{BARG}}$.
- Set $j = 0, x_0 = x, \pi_0 = \pi$.
- Parse π_0 as (z_0, π'_0).
- While $z_j \neq \emptyset$:
  - Set $j = j + 1$.
  - Extract $(x_j, \rho_j, \pi_j) \leftarrow \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}_{\mathsf{BARG}}, \pi'_{j-1})$
  - Parse π_j as (z_j, π'_j).
- Output (x'_j, π'_j).

4.2.4 Analysis.

Completeness and merge completeness follow directly from the underlying building blocks. Merge efficiency for circuits follows by efficiency of the one-hop argument system. Trapdoor hiding follows from trapdoor hiding of the OneHop argument system, and of the BARG.

We now turn to prove trapdoor extraction. As in Theorem 4.1, it suffices to prove the following claim:

Claim 4.9. *Let $t(\cdot)$ be a polynomial. Let $\mathcal{A}$ be a poly-size adversary. Let (x, π) be the output of $\mathcal{A}$. Let $j^* \geq 0$ be the number of iterations performed by the extractor until it halts. For $j \in \mathbb{N}$, denote by A_j the event that the extractor did not halt in the first j iterations, and at the beginning of the $j + 1$ iteration of the extraction algorithm, the following holds:*

$$x_j \notin L^*_{\tau(\beta)} \ ,$$
$$\mathsf{Verify}(\beta, \mathsf{pk}, x_j, \pi_j) = 1 \ .$$

Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any polynomial $j(\cdot)$ such that $1 \leq j(\lambda) \leq t(\lambda)$ for any $\lambda \in \mathbb{N}$, and for any mapping $\tau : \mathrm{B}_\lambda \to \mathrm{T}_\lambda$ that is sound for IM, it holds that for any $\lambda \in \mathbb{N}$,

$$\Pr\left[\bar{A}_{j(\lambda)} \wedge A_{j(\lambda)-1} \wedge j^* \geq j(\lambda)\right] \leq \mathsf{negl}(\lambda) \ .$$

Proof. We begin by applying strong somewhere extraction of the BARG.

Let $(\mathsf{pk}_{\mathsf{OneHop}}, \mathsf{td}_{\mathsf{OneHop}}) \leftarrow \mathsf{Gen}_{\mathsf{OneHop}}(1^\lambda)$, and consider the following adversary $\mathcal{A}'[j, \mathsf{pk}_{\mathsf{OneHop}}]$ to the BARG scheme, which does the following given $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}})$:

- Set $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}})$.
- Run the original adversary and obtain $(x, \pi) \leftarrow \mathcal{A}(\mathsf{pk})$.

- Run the extraction algorithm $\mathsf{Extract}(\mathsf{td}_{\mathsf{BARG}}, x, \pi)$ for $j-1$ iterations, and obtain $(x_{j-1}, \rho_{j-1}, \pi_{j-1})$, where $\pi_{j-1} = (z_{j-1}, \pi'_{j-1})$.
- If $z_{j-1} = \emptyset$, abort. Otherwise, parse $z_{j-1} = (\pi_{\mathsf{OneHop}}, k)$.
- Construct the machine $\mathcal{M} := \mathcal{M}[\mathsf{pk}, \beta, z_{j-1}, x_{j-1}]$ as in Verify: given (i, x_i, ρ_i, π^*), it accepts if all all the following checks pass:
  - Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{OneHop}})$.
  - Check that $\mathsf{Verify}(\beta, \mathsf{pk}, x_i, \pi^*) = 1$.
  - Parse $z_{j-1} = (\pi_{\mathsf{OneHop}}, k)$, and check that $\mathsf{Verify}_{\mathsf{OneHop}}(\beta, \mathsf{pk}_{\mathsf{OneHop}}, x_{j-1}, \pi_{\mathsf{OneHop}}, k, i, x_i, \rho_i)$.
- Output $(z_{j-1}, \mathcal{M}, k, \pi'_{j-1})$.

Assuming that $A_{j-1} \wedge j \leq j^*$, we have that $\mathcal{A}'[j, \mathsf{pk}_{\mathsf{OneHop}}]$ does not abort. Furthermore, note that A_{j-1} implies $\mathsf{Verify}(\beta, \mathsf{pk}, x_{j-1}, \pi_{j-1}) = 1$, which implies $\mathsf{Verify}_{\mathsf{BARG}}(\mathsf{pk}_{\mathsf{BARG}}, z_{j-1}, (\mathcal{M}, k), \pi'_{j-1}) = 1$. Therefore, by BARG strong somewhere extraction, we get that except with negligible probability, the extracted witness $w_j = (x_j, \rho_j, \pi_j)$ in the jth iteration satisfies $\mathcal{M}(f(z_{j-1}), w_j) = 1$. By definition of $\mathcal{M}$, this implies that $\mathsf{Verify}(\beta, \mathsf{pk}, x_j, \pi_j) = 1$.

Now all that is left to prove is that $x_j \notin L^*_\tau$. For this we use strong trapdoor extraction of the OneHop argument system. Consider the adversary $\mathcal{A}'_j$ which does the following given $(\mathsf{pk}_{\mathsf{OneHop}}, \mathsf{td}_{\mathsf{OneHop}})$:

- Let $f = \mathcal{F}[\mathsf{td}_{\mathsf{OneHop}}]$ be the machine that on input π_{OneHop}, k, outputs $\mathsf{Extract}(\mathsf{td}_{\mathsf{OneHop}}, k, \pi_{\mathsf{OneHop}})$.
- Generate $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}_{\mathsf{BARG}}(1^\lambda, f)$.
- Run $\mathcal{A}'[j, \mathsf{pk}_{\mathsf{OneHop}}]$, and keep intermediate variables $x_{j-1}, z_{j-1} = (\pi_{\mathsf{OneHop}}, k)$.
- Run the extraction algorithm for an additional jth iteration, and obtain (x_j, ρ_j, π_j).
- Extract $j^* = \mathsf{Extract}(\mathsf{td}_{\mathsf{OneHop}}, k, \pi_{\mathsf{OneHop}})$.
- Output $(x_{j-1}, \pi_{\mathsf{OneHop}}, k, j^*, x_j, \rho_j)$.

By strong trapdoor extraction, we have

$$\Pr\left[\begin{array}{l|l} x_{j-1} \notin L^*_{\tau(\beta)} \ \wedge \ j = j^* \wedge & \\ \mathsf{Verify}_{\mathsf{OneHop}}(\beta, \mathsf{pk}, x, \pi, j, x_j, \rho_j) = 1 \ \rightarrow \ x_j \notin L^*_{\tau(\beta)} & \begin{array}{l}(\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, \tau(\beta)) \\ (x_{j-1}, \pi, j, x_j, \rho_j) \leftarrow \mathcal{A}'_j(\mathsf{pk}, \mathsf{td}) \\ j^* \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)\end{array} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Observe that since we assumed the event A_{j-1}, we already have $x_j \notin L^*_{\tau(\beta)}$. We also have $j = j^*$ by definition of $\mathcal{A}'_j$. Finally, since we have $\mathcal{M}(f(z_{j-1}), w_j) = 1$ (this followed from strong BARG somewhere extraction), by definition of $\mathcal{M}$ we get

$$\mathsf{Verify}_{\mathsf{OneHop}}(\beta, \mathsf{pk}, x, \pi, f(z_{j-1}), x_j, \rho_j) = 1 \ .$$

By definition of $f = \mathcal{F}[\mathsf{td}_{\mathsf{OneHop}}]$, we have $f(z_{j-1}) = \mathsf{Extract}(\mathsf{td}_{\mathsf{OneHop}}, k, \pi_{\mathsf{OneHop}}) = j^* = j$. Thus the left side of the implication holds, and we get that except with negligible probability, $x_j \notin L^*_{\tau(\beta)}$, which concludes the proof.

□

4.3 Universal Soundness

Claim 4.10. *Assume that for any language families $L \in \mathsf{NP}$ an $L^* \in P$ and instance merger IM, there exists a mergeable argument system Π for (L, L^*, IM). Then, for any $L \in \mathsf{NP}, L^* \in P$ and instance merger IM, there exists a mergeable argument Π' for (L, L^*, IM) with universal soundness.*

Furthermore, if Π has trapdoor extraction, then Π' has universal trapdoor extraction, and if Π has merge efficiency for circuits, then the same holds for Π'.

Proof. We alter the proof system as follows. Given a language L_β, define the extended language $\widehat{L}_\beta = \{(t, x) \mid t \in \mathbb{N} \wedge x \in L_\beta\}$. Next, define an extended instance merger $\widehat{\mathsf{IM}}$ such that given extended instances $\hat{x}_1 = (t_1, x_1), \ldots, \hat{x}_k = (t_1, x_1)$, computes

$$\widehat{\mathsf{IM}}(\beta, \hat{x}_1, \ldots, \hat{x}_k) = (\max_i(t_i) + 1, \mathsf{IM}(\beta, x_1, \ldots, x_k)) \ .$$

The instance merger outputs $\perp$ if one of the instances is not in the form (t, x) [7]. Also define the weight function $\widehat{\mathsf{W}}(t, x)) = t$. Overall, this induces extended language families and extended instance merger $(\widehat{L}, \widehat{L^*}, \widehat{\mathsf{IM}})$. Moreover, this is a family of trapdoor languages. Let $\widehat{\Pi} = (\widehat{\mathsf{Gen}}, \widehat{\mathsf{Prove}}, \widehat{\mathsf{Verify}}, \widehat{\mathsf{PM}})$ be a mergeable argument system for $(\widehat{L}, \widehat{L^*}, \widehat{\mathsf{IM}})$.

We now describe Π'. A proof π' of an instance x is of the form (t, π) where t is a weight and π is a proof in the original mergeable proof system for the extended instance (t, x), padded to length $\max(|\pi|, t)$. When verifying a proof (t, π), we check that indeed $|\pi| \geq t$. Therefore, an adversary $\mathcal{A}$ of size at most T cannot produce an accepting proof (t, π) for a statement x such that $t > T$. Therefore the extended instance (t, x) has weight bounded by the polynomial T, and we can invoke soundness. Similarly, we can also invoke trapdoor extraction (assuming it holds for Π), and note that the extractor's running time is polynomial in its input length, since it is guaranteed to be polynomial in its input length and the weight t, and its input contains the new proof of length $\geq t$.

Moreover, this transformation preserves merge efficiency: given instance-proof pairs $(x_1, \pi_1), \ldots, (x_k, \pi_k)$ and merged (x, π), and proofs $\pi'_1, \ldots, \pi'_k, \pi'$ in the new system, where we write $\pi'_i = (t_i, \pi_i)$ and $\pi' = (t, \pi)$, since we have $t = \max_i t_i + 1$, we get that

$$\begin{aligned}
|\pi'| &= \max(|\pi|, t) \\
&\leq \max(|(x_1, \ldots, x_k)| \cdot \mathsf{poly}(\lambda) + \max_i |\pi_i|, t) \\
&\leq |(x_1, \ldots, x_k)| \cdot \mathsf{poly}(\lambda) + \max(\max_i |\pi_i|, \max_i t_i) \\
&= |(x_1, \ldots, x_k)| \cdot \mathsf{poly}(\lambda) + \max_i(\max(|\pi_i|, t_i)) \\
&= |(x_1, \ldots, x_k)| \cdot \mathsf{poly}(\lambda) + \max_i |\pi'_i| \ ,
\end{aligned}$$

as required. The same derivation shows that merge efficiency for circuits is preserved. □

5 Mergeable Arguments from Indistinguishability Obfuscation

In this section we describe our construction of mergeable arguments based on indistinguishability obfuscation $i\mathcal{O}$, puncturable pseudorandom function PPRF and injective PRG G. Let $L \in \mathsf{NP}$ and $L^* \in P$ be language families (recalling the terminology from Section 3). Let V_L be the verifier of L, and let D_{L^*} be the decider

[7] As usual, this is encoded in a self-delimiting manner.

of L^*. Let IM be an instance merger for L, and let W be a weight function. The construction generalizes the technique presented by [DJJ+25] to construct incrementally verifiable computation (IVC) for trapdoor languages. We adapt this technique to facilitate merging two proofs instead of updating a single proof (see Section 5.3 below for a more detailed comparison with [DJJ+25], and Remark 5.1 for merging more than two instances at a time). In the following we start with specifying the parameters of the construction.

5.1 Parameters

Let λ be the security parameter, n be the instance length, and let p be a polynomial such that $\mathsf{W}(x)$ can be described with $p(|x|)$ bits. Let $m(\lambda, n)$ be a polynomial bound on the on the witnesses length [8]. Assume that:

- $i\mathcal{O}$ is a $(2^{\lambda^{\delta_{i\mathcal{O}}}}, 2^{-\lambda^{\delta_{i\mathcal{O}}}})$-secure indistinguishability obfuscator for some constant $\delta_{i\mathcal{O}}$.
- PPRF is a $(2^{\lambda^{\delta_{\mathsf{PPRF}}}}, 2^{-\lambda^{\delta_{\mathsf{PPRF}}}})$-secure puncturable pseudorandom function for some constant δ_{PPRF}.
- There exists an injective $(2^{\lambda^{\delta_{\mathsf{PRG}}}}, 2^{-\lambda^{\delta_{\mathsf{PRG}}}})$-secure pseudorandom generator for some constant δ_{PRG}. If polynomially secure pseudorandom generator used in the analysis, set in the following $\delta_{\mathsf{PRG}} = 1$.

We set the security parameters used by the scheme as follows:

- $\lambda_{\mathsf{PRG}} = (2\lambda)^{1/\delta_{\mathsf{PRG}}}$.
- $\lambda_{\mathsf{PPRF}} = (2\lambda + n)^{1/\delta_{\mathsf{PPRF}}}$.
- $\lambda_{i\mathcal{O}} = (2\lambda + n)^{1/\delta_{i\mathcal{O}}}$.

It follows that the distinguishing advantages corresponding to the PPRF and $i\mathcal{O}$ schemes are $\varepsilon_{\mathsf{PPRF}} = \varepsilon_{i\mathcal{O}} = 2^{-(2\lambda+n)}$. The distinguishing advantge correpsonding to the PRG is $\varepsilon_{\mathsf{PRG}} = 2^{-2\lambda}$ for a subexponentially secure PRG, and some negligible function $\mathsf{negl}(\lambda)$ if polynomially secure PRG is used in the analysis.

Next, The input and output of the pseudorandom functions and pseudorandom generator used by the scheme is described in the following table:

Primitive	Security Parameter	Input Size	Output Size
PRG	λ_{PRG}	λ_{PRG}	$2\lambda_{\mathsf{PRG}}$
PPRF scheme with the key K	λ_{PPRF}	n	λ_{PRG}
PPRF scheme with the key K_{off}	λ_{PPRF}	n	λ_{PRG}
PPRF scheme with the key K_{chal}	λ_{PPRF}	$p(n)$	λ_{PRG}

Table 1: Parameters of the pseudorandom functions and pseudorandom generator

The circuits PM and Verify are constructed to accept a proof π of size $\ell_\pi = \max(\lambda_{\mathsf{PRG}}, m(\lambda, n))$, i.e. the maximum size of the output of a PPRF evaluation and a witness length [9]. Let $\ell_{i\mathcal{O}} = \mathsf{poly}(\lambda, n)$ be the

[8] Recall that L in NP.

[9] The circuits PM, Verify parse the proof π accordingly as either a witness or PPRF evaluation.

maximum size of any circuit PM, Verify used by the construction or in the hybrids in the security analysis in Section 5.3[10].
In the following we present the construction.

5.2 Construction

The construction of Π = (Gen, Prove, Verify, PM) is as follows:

Gen : On input 1^λ and instance bound 1^n, the generation algorithm is as follows:

- Set adjusted security parameters $\lambda_{iO} = p_{iO}(\lambda, n)$, $\lambda_{\mathsf{PPRF}} = p_{\mathsf{PPRF}}(\lambda, n)$, $\lambda_{\mathsf{PRG}} = p_{\mathsf{PRG}}(\lambda, n)$ for polynomials $p_{iO}, p_{\mathsf{PPRF}}, p_{\mathsf{PRG}}$ specified in Section 5.1.
- Sample PPRF seed $K \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
- Let $\mathsf{PM}[K]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{PM}} \leftarrow i\mathcal{O}(1^{\lambda_{iO}}, \mathsf{PM}[K])$.
- Let $\mathsf{Verify}[K]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{Verify}} \leftarrow i\mathcal{O}(1^{\lambda_{iO}}, \mathsf{Verify}[K])$.
- Output $(\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$.

Prove: Output w.

Verify: On input public parameters β, public key pk, instance x, and proof π, the verifier algorithm parses $\mathsf{pk} = (\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$ and returns $\widehat{\mathsf{Verify}}(\beta, x, \pi)$.

PM: On input public parameters β, public key pk, and pairs $(x_0, \pi_0), (x_1, \pi_1)$, the proof merger algorithm parses $\mathsf{pk} = (\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$ and returns as $\widehat{\mathsf{PM}}(\beta, ((x_0, \pi_0), (x_1, \pi_1)))$.

We now describe the circuits $\mathsf{PM}[K]$ and $\mathsf{Verify}[K]$ used above. The circuits are padded to size $\ell_{iO}(\lambda, n)$ where the function ℓ_{iO} is fixed in the analysis.

$\mathsf{PM}[K]$: On input public parameters β, and pairs $(x_0, \pi_0), (x_1, \pi_1)$ such that $x_i \in \{0,1\}^n$ and $\pi_i \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- Compute merged instance: $x' = \mathsf{IM}(\beta, x_0, x_1)$.
- If $x' = \perp$, return $\perp$.
- Verify previous proofs: For $i \in \{0,1\}$:
  - If $\mathsf{V}_L(\beta, x_i, \pi_i) \neq 1$ and $\mathsf{PPRF.Eval}_K(x_i) \neq \pi_i$, return $\perp$.
- Compute new proof: $\pi' = \mathsf{PPRF.Eval}_K(x')$.
- Return π'.

$\mathsf{Verify}[K]$: On input public parameters β, instance $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- If $\mathsf{V}_L(\beta, x, \pi) = 1$ or $\mathsf{PPRF.Eval}_K(x) = \pi$, return 1.
- Else, return 0.

[10] Note that ℓ_{iO} depends in particular on the size of the circuit G computing the PRG, the size of the decider Dec, and the parameters $\beta, \tau(\beta)$, which have polynomial size in the security parameter λ.

Remark 5.1 (Merging more than two instances at a time)**.** For simplicity, the construction above is written to support merging of two instances at a time. In order to support merging $k > 2$ at a time, the circuit PM can be altered in a natural way to accept k instances and proofs. In this case, the parameter k should be provided as an additional parameter to Gen, which generates the circuit PM. In particular, the same security analysis applies.

Theorem 5.1. *Assuming:*

- $i\mathcal{O}$ *is subexponentially secure indistinguishability obfuscator for some constant* $\delta_{i\mathcal{O}}$*.*
- PPRF *is subexponentially secure puncturable pseudorandom function for some constant* δ_{PPRF}*.*
- *There exists an injective PRG.*

Then Π *is depth-compact mergeable argument for* (L, L^*, IM)*. Additionally, if there exists a subexponentially secure injective PRG, then* Π *has universal soundness.*

Remark 5.2. As in [DJJ+25], the injective PRG does not appear in the construction, and only appears in the security analysis. In particular, Gen is only required to know a bound on the circuit size required to compute G.

Corollary 5.2. *Assuming:*

- *There exists subexponentially secure indistinguishability obfuscation.*
- *There exists subexponentially secure one-way function.*

Then there exists a depth-compact mergeable argument for (L, L^*, IM) *with universal soundness.*

Proof. By [BPW16], it follows that an injective pseudorandom generator exists assuming subexponentially secure indistinguishability obfuscation and subexponentially secure one-way function. Moreover, by [GGM86, SW14], it follows that subexponentially secure puncturable pseudorandom function can be constructed from subexponentially secure one-way function. □

Theorem 5.1 follows from the following claims:

Claim 5.3. Π *has completeness, merge completeness, and depth-compactness.*

Proof. Completeness follows from the completeness of V_L and the correctness of the $i\mathcal{O}$ scheme. Merge completeness follows from the correctness of the $i\mathcal{O}$ scheme. For depth compactness, let $\ell(\lambda)$ be a bound on the witness length for L, for some polynomial ℓ. Let $\ell' \geq \ell$ be the output length of PPRF. Finally, observe that a merged proof is length $\ell'(\lambda)$. □

Claim 5.4. *Assuming:*

- $i\mathcal{O}$ *is a* $(2^{\lambda^{\delta_{i\mathcal{O}}}}, 2^{-\lambda^{\delta_{i\mathcal{O}}}})$*-secure indistinguishability obfuscator for some constant* $\delta_{i\mathcal{O}}$*.*
- PPRF *is a* $(2^{\lambda^{\delta_{\mathsf{PPRF}}}}, 2^{-\lambda^{\delta_{\mathsf{PPRF}}}})$*-secure puncturable pseudorandom function for some constant* δ_{PPRF}*.*
- *There exists an injective PRG.*

Then Π *is sound. Additionally, if there exists a subexponentially secure injective PRG, then* Π *has universal soundness.*

In the following we sketch the security analysis.

5.3 Soundness

For the analysis of our scheme, we use several claims that were proven in [DJJ+25]. We can do this because the general structure of our construction is "isomorphic" to that of [DJJ+25]. We now explain the mapping between their terminology and ours so that we can point to their claims throughout the analysis.

Given a circuit C, the IVC language L^C_{IVC} consists of statements of the form $\hat{x} = (x_0, x_i, i)$ where there exist $x_1, \ldots, x_{i-1}$ and witnesses $w_1, \ldots w_i$ such that $C(x_{j-1}, w_j) = x_j$ for every $1 \leq j \leq i$. We observe that an IVC scheme is a special case of mergeable argument scheme. In more detail, we can consider an IVC scheme in terms of a mergeable argument scheme:

- The public parameter β is simply the circuit C. Note that in the IVC scheme of [DJJ+25], the circuit C is given to Gen, whereas in our definition of a mergeable argument scheme the public parameters is not given to Gen. However, this does not effect the soundness analysis, since β cannot be chosen adaptively by the adversary.
- The instance merger IM has arity one, and accepts an auxiliary input w (see Remark 3.1 for further details regarding instance merger with auxiliary input). On input $\hat{x} = (x_0, x_i, i)$ and axillary input w, the instance merger computes $\hat{x}' = (x_0, C(x_i, w), i + 1)$.
- The weight of an instance $\hat{x} = (x_0, x_i, i)$ is the step-counter i, whereas in mergeable argument scheme the weight function W can be an arbitrary poly-size computable function. Importantly, given instances x_0, x_1 and their merged instance $x' = \mathsf{IM}(\beta, (x_0, x_1))$, if $x' \neq \perp$, then $\mathsf{W}(x') \geq \max_i(\mathsf{W}(x_i)) + 1$. The monotone weight increase allows us to follow the security proof of [DJJ+25], and argue inductively that if an adversary cannot produce a faulty instance with weight at most W with an accepting proof, it cannot do that also for $W + 1$.
- The trapdoor considered in [DJJ+25] allows to decide the IVC language, i.e. allows to determine if $\hat{x} = (x_0, x_i, i) \in L^C_{\mathrm{IVC}}$. In our construction we consider the general case of trapdoor languages.

We sketch below the main hybrids. Let $L = \{L_\beta\}_{\beta \in \mathrm{B}}$ and $L^* = \{L^*_\tau\}_{\tau \in \mathrm{T}}$ be language families. Let $\bar{\mathrm{W}}$ be a weight bound, and let $\beta \in \mathrm{B}$ be the public parameters. Let $\mathcal{A}$ be a poly-size adversary. Let $\tau : \mathrm{B} \to \mathrm{T}$ be a mapping that is sound for L and for IM. Let $\mathsf{Dec}_{\tau(\beta)}(\cdot) = \mathsf{Dec}(\tau(\beta), \cdot)$ be the decider of $L^*_{\tau(\beta)}$. Let G be an injective pseudorandom generator $G : \{0,1\}^{\lambda_{\mathrm{PRG}}} \to \{0,1\}^{2\lambda_{\mathrm{PRG}}}$. Finally, let p_i be the probability that the output of the i-th hybrid is 1. In the following hybrids only Gen and helper circuits PM, Verify are changed. We consider the following main hybrids:

$\mathcal{H}_0$: This corresponds to the real experiment.

- $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- $(x, \pi) \leftarrow \mathcal{A}(\mathsf{pk})$.
- Output 1 if $\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1$ and $x \notin L^*_{\tau(\beta)}$ and $\mathsf{W}(x) \leq \bar{\mathrm{W}}$. Otherwise, output 0.

$\mathcal{H}_1$: Make the hybrids efficient.

- $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.

- $(x, \pi) \leftarrow \mathcal{A}(\mathsf{pk})$.
- Output 1 if $\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi) = 1$ and $x \notin L^*_{\tau(\beta)}$ and $\mathsf{Dec}_{\tau(\beta)}(\cdot)(x) = 1$ and $\mathsf{W}(x) \leq \bar{\mathsf{W}}$. Otherwise, output 0.

In the next set of hybrids we plant a challenge in the proof for all faulty statements $x \notin L^*_{\tau(\beta)}$ with weight at most $\bar{\mathsf{W}}$. This is done in lexicographical order, one instance at a time. In more detail, in hybrid x^* the proof for any faulty instance $x \notin L^*_{\tau(\beta)}$ such that $x \leq x^*$ is switched to $\pi = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x))$, for fresh seeds $K_{\text{off}}, K_{\text{chal}}$ sampled by Gen.

$\mathcal{H}_2^{x^*}$: Plant a challenge in the proof for any faulty instance $x \notin L^*_{\tau(\beta)}$ such that $x \leq x^*$:

Gen : On input 1^λ and instance bound 1^n, the generation algorithm is as follows:

- Set adjusted security parameters $\lambda_{iO} = p_{iO}(\lambda, n)$, $\lambda_{\mathsf{PPRF}} = p_{\mathsf{PPRF}}(\lambda, n)$, $\lambda_{\mathsf{PRG}} = p_{\mathsf{PRG}}(\lambda, n)$ for polynomials $p_{iO}, p_{\mathsf{PPRF}}, p_{\mathsf{PRG}}$ specified in Section 5.1.
- Sample PPRF seeds:
  - $K \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{off}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{chal}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
- Let $\mathsf{PM}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}, x^*]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{PM}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{PM}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}, x^*])$.
- Let $\mathsf{Verify}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}, x^*]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{Verify}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{Verify}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}, x^*])$.
- Output $(\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$.

$\mathsf{PM}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}, x^*]$: On input public parameters β, and pairs $(x_0, \pi_0), (x_1, \pi_1)$ such that $x_i \in \{0,1\}^n$ and $\pi_i \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- Compute merged instance: $x' = \mathsf{IM}(\beta, x_0, x_1)$.
- If $x' = \perp$, return $\perp$.
- Verify previous proofs: For $i \in \{0,1\}$:
  - If $x_i \leq x^*$ and $\mathsf{Dec}_{\tau(\beta)}(x_i) = 0$
    - If $\pi_i \neq \mathsf{PPRF.Eval}_{K_{\text{off}}}(x_i) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_i))$, return $\perp$
  - Else, if $\mathsf{V}_L(\beta, x_i, \pi_i) \neq 1$ and $\mathsf{PPRF.Eval}_K(x_i) \neq \pi_i$, return $\perp$.
- Compute new proof:
  - If $x' \leq x$ and $\mathsf{Dec}_{\tau(\beta)}(x') = 0$:
    - $\pi' = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x') \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x'))$
  - Else, $\pi' = \mathsf{PPRF.Eval}_K(x')$.
- Return π'.

$\mathsf{Verify}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}, x^*]$: On input public parameters β, instance $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- If $x \leq x^*$ and $\mathsf{Dec}_{\tau(\beta)}(x) = 0$:
  - If $\pi \neq \mathsf{PPRF.Eval}_{K_{\text{off}}}(x) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x))$, return 0.
- Else, if $\mathsf{V}_L(\beta, x, \pi) \neq 1$ and $\mathsf{PPRF.Eval}_K(x) \neq \pi$, return 0.
- Else, return 1.

The next hybrid is obtained after planting a challenge in the proof of any faulty instance $x \notin L^*_{\tau(\beta)}$.

$\mathcal{H}_3$: Plant a challenge in the proof of any faulty instance $x \notin L^*_{\tau(\beta)}$:

Gen : On input 1^λ and instance bound 1^n, the generation algorithm is as follows:

- Set adjusted security parameters $\lambda_{iO} = p_{iO}(\lambda, n)$, $\lambda_{\mathsf{PPRF}} = p_{\mathsf{PPRF}}(\lambda, n)$, $\lambda_{\mathsf{PRG}} = p_{\mathsf{PRG}}(\lambda, n)$ for polynomials $p_{iO}, p_{\mathsf{PPRF}}, p_{\mathsf{PRG}}$ specified in Section 5.1.
- Sample PPRF seeds:
  - $K \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{off}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{chal}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
- Let $\mathsf{PM}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{PM}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{PM}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Let $\mathsf{Verify}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{Verify}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{Verify}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Output $(\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$.

$\mathsf{PM}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, and pairs $(x_0, \pi_0), (x_1, \pi_1)$ such that $x_i \in \{0,1\}^n$ and $\pi_i \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- Compute merged instance: $x' = \mathsf{IM}(\beta, x_0, x_1)$.
- If $x' = \perp$, return $\perp$.
- Verify previous proofs: For $i \in \{0,1\}$:
  - If $\mathsf{Dec}_{\tau(\beta)}(x_i) = 0$:
    - If $\pi_i \neq \mathsf{PPRF.Eval}_{K_{\text{off}}}(x_i) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_i))$, return $\perp$.
  - Else, if $\mathsf{V}_L(\beta, x_i, \pi_i) \neq 1$ and $\mathsf{PPRF.Eval}_K(x_i) \neq \pi_i$, return $\perp$.
- Compute new proof:
  - If $\mathsf{Dec}_{\tau(\beta)}(x') = 0$:
    - $\pi' = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x') \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x'))$.
  - Else, $\pi' = \mathsf{PPRF.Eval}_K(x')$.
- Return π'.

$\mathsf{Verify}[\mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, instance $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- If $\mathsf{Dec}_{\tau(\beta)}(x) = 0$:
  - If $\pi \neq \mathsf{PPRF.Eval}_{K_{\text{off}}}(x) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x))$, return 0.
- Else, if $\mathsf{V}_L(\beta, x, \pi) \neq 1$ and $\mathsf{PPRF.Eval}_K(x) \neq \pi$, return 0.
- Else, return 1.

Next we apply the injective pseudorandom function G.

$\mathcal{H}_4$: Apply the injective PRG G:

Gen : On input 1^λ and instance bound 1^n, the generation algorithm is as follows:

- Set adjusted security parameters $\lambda_{iO} = p_{iO}(\lambda, n)$, $\lambda_{\mathsf{PPRF}} = p_{\mathsf{PPRF}}(\lambda, n)$, $\lambda_{\mathsf{PRG}} = p_{\mathsf{PRG}}(\lambda, n)$ for polynomials $p_{iO}, p_{\mathsf{PPRF}}, p_{\mathsf{PRG}}$ specified in Section 5.1.
- Sample PPRF seeds:
  - $K \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{off}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{chal}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
- Let $\mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{PM}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Let $\mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{Verify}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Output $(\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$.

$\mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, and pairs $(x_0, \pi_0), (x_1, \pi_1)$ such that $x_i \in \{0,1\}^n$ and $\pi_i \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- Compute merged instance: $x' = \mathsf{IM}(\beta, x_0, x_1)$.
- If $x' = \perp$, return $\perp$.
- Verify previous proofs: For $i \in \{0,1\}$:
  - If $\mathsf{Dec}_{\tau(\beta)}(x_i) = 0$:
    - If $G(\pi_i \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x_i)) \neq G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_i)))$, return $\perp$
  - Else, if $\mathsf{V}_L(\beta, x_i, \pi_i) \neq 1$ and $\mathsf{PPRF.Eval}_K(x_i) \neq \pi_i$, return $\perp$.
- Compute new proof:
  - If $\mathsf{Dec}_{\tau(\beta)}(x') = 0$:
    - $\pi' = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x') \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x'))$.
  - Else, $\pi' = \mathsf{PPRF.Eval}_K(x')$.
- Return π'.

$\mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, instance $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- If $\mathsf{Dec}_{\tau(\beta)}(x) = 0$:
  - If $G(\pi \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x)) \neq G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x)))$, return 0.
- Else, if $\mathsf{V}_L(\beta, x, \pi) \neq 1$ and $\mathsf{PPRF.Eval}_K(x) \neq \pi$, return 0.
- Else, return 1.

In the next set of hybrids we puncture out the proofs for every faulty instance $x \notin L^*_{\tau(\beta)}$ with weight at most W^*, where $\mathsf{W}^* \leq \bar{\mathsf{W}}$. In more detail, in hybrid W^*, no proof for faulty instance $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \leq \mathsf{W}^*$ is produced or accepted.

$\mathcal{H}_5^{\mathsf{W}^*}$: Remove proof for any faulty instance $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \leq \mathsf{W}^*$:

Gen : On input 1^λ and instance bound 1^n, the generation algorithm is as follows:

- Set adjusted security parameters $\lambda_{iO} = p_{iO}(\lambda, n)$, $\lambda_{\mathsf{PPRF}} = p_{\mathsf{PPRF}}(\lambda, n)$, $\lambda_{\mathsf{PRG}} = p_{\mathsf{PRG}}(\lambda, n)$ for polynomials $p_{iO}, p_{\mathsf{PPRF}}, p_{\mathsf{PRG}}$ specified in Section 5.1.
- Sample PPRF seeds:
  - $K \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{off}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{chal}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
- Let $\mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{PM}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Let $\mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{Verify}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Output $(\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$.

$\mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, and pairs $(x_0, \pi_0), (x_1, \pi_1)$ such that $x_i \in \{0,1\}^n$ and $\pi_i \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- Compute merged instance: $x' = \mathsf{IM}(\beta, x_0, x_1)$.
- If $x' = \perp$, return $\perp$.
- Verify previous proofs: For $i \in \{0,1\}$:
  - If $\mathsf{Dec}_{\tau(\beta)}(x_i) = 0$:
    - If $\mathsf{W}(x_i) \leq \mathsf{W}^*$, return $\perp$.
    - Else, if $G(\pi \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x_i)) \neq G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_i)))$, return $\perp$.
  - Else, if $\mathsf{V}_L(\beta, x_i, \pi_i) \neq 1$ and $\mathsf{PPRF.Eval}_K(x_i) \neq \pi_i$, return $\perp$.
- Compute new proof:
  - If $\mathsf{Dec}_{\tau(\beta)}(x') = 0$:

* If $\mathsf{W}(x') \le \mathsf{W}^*$, return $\bot$.
* Else, $\pi' = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x') \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x'))$.

– Else, $\pi' = \mathsf{PPRF.Eval}_K(x')$.

- Return π'.

$\mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, instance $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- If $\mathsf{Dec}_{\tau(\beta)}(x) = 0$:
  - If $\mathsf{W}(x) \le \mathsf{W}^*$, return 0.
  - Else, if $G(\pi \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x)) \neq G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x)))$, return 0.
- Else, if $\mathsf{V}_L(\beta, x, \pi) \neq 1$ and $\mathsf{PPRF.Eval}_K(x) \neq \pi$, return 0.
- Else, return 1.

The next hybrid is a sub-hybrid between $\mathcal{H}_5^{\mathsf{W}^*}$ and $\mathcal{H}_5^{\mathsf{W}^*+1}$. In this sub-hybrid we no longer generate proof for any faulty instance $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \le \mathsf{W}^* + 1$.

$\mathcal{H}_{5,1}^{\mathsf{W}^*}$: Stop generating proof for any faulty instance $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \le \mathsf{W}^* + 1$:

Gen : On input 1^λ and instance bound 1^n, the generation algorithm is as follows:

- Set adjusted security parameters $\lambda_{iO} = p_{iO}(\lambda, n)$, $\lambda_{\mathsf{PPRF}} = p_{\mathsf{PPRF}}(\lambda, n)$, $\lambda_{\mathsf{PRG}} = p_{\mathsf{PRG}}(\lambda, n)$ for polynomials $p_{iO}, p_{\mathsf{PPRF}}, p_{\mathsf{PRG}}$ specified in Section 5.1.
- Sample PPRF seeds:
  - $K \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{off}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
  - $K_{\text{chal}} \leftarrow \{0,1\}^{\lambda_{\mathsf{PPRF}}}$.
- Let $\mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{PM}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Let $\mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}]$ be the circuit specified below, padded to size ℓ_{iO}. Sample $\widehat{\mathsf{Verify}} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\text{off}}, K_{\text{chal}}])$.
- Output $(\widehat{\mathsf{PM}}, \widehat{\mathsf{Verify}})$.

$\mathsf{PM}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, and pairs $(x_0, \pi_0), (x_1, \pi_1)$ such that $x_i \in \{0,1\}^n$ and $\pi_i \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- Compute merged instance: $x' = \mathsf{IM}(\beta, x_0, x_1)$.
- If $x' = \bot$, return $\bot$.
- Verify previous proofs: For $i \in \{0,1\}$:
  - If $\mathsf{Dec}_{\tau(\beta)}(x_i) = 0$:
    * If $\mathsf{W}(x_i) \le \mathsf{W}^*$, return $\bot$.

* Else, if $G(\pi \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x_i)) \neq G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_i)))$, return $\perp$.
  - Else, if $\mathsf{V}_L(\beta, x_i, \pi_i) \neq 1$ and $\mathsf{PPRF.Eval}_K(x_i) \neq \pi_i$, return $\perp$.
- Compute new proof:
  - If $\mathsf{Dec}_{\tau(\beta)}(x') = 0$:
    * If $\mathsf{W}(x') \leq \mathsf{W}^* + 1$, return $\perp$.
    * Else, $\pi' = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x') \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x'))$.
  - Else, $\pi' = \mathsf{PPRF.Eval}_K(x')$.
- Return π'.

$\mathsf{Verify}[G, \mathsf{Dec}_{\tau(\beta)}, K, K_{\mathbf{off}}, K_{\mathbf{chal}}]$: On input public parameters β, instance $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{\ell_\pi}$, the circuit computes:

- If $\mathsf{Dec}_{\tau(\beta)}(x) = 0$:
  - If $\mathsf{W}(x) \leq \mathsf{W}^*$, return 0.
  - Else, if $G(\pi \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x)) \neq G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x)))$, return 0.
- Else, if $\mathsf{V}_L(\beta, x, \pi) \neq 1$ and $\mathsf{PPRF.Eval}_K(x) \neq \pi$, return 0.
- Else, return 1.

Indistinguishability between the hybrids is proved by the following lemmas:

Lemma 5.5. *Let p_0 and p_1 be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$. Then $p_0 = p_1$.*

Lemma 5.6. *Let x_{first} be the first instance according to the lexicographical order. And let p_1 and $p_{2,first}$ be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_1$ and $\mathcal{H}_2^{x_{first}}$. Then there exist a constant $c_{1,2}$ such that $\left|p_1 - p_{2,first}\right| \leq c_{1,2} \cdot (\varepsilon_{\mathsf{PPRF}} + \varepsilon_{iO})$.*

Lemma 5.7. *Let x_i and x_{i+1} be two consecutive instances according to the lexicographical order. And let p_{2,x_i} and $p_{2,x_{i+1}}$ be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_2^{x_i}$ and $\mathcal{H}_2^{x_{i+1}}$. Then there exist a constant c_2 such that $\left|p_{2,x_i} - p_{2,x_{i+1}}\right| \leq c_2 \cdot (\varepsilon_{\mathsf{PPRF}} + \varepsilon_{iO})$.*

Lemma 5.8. *Let x_{final} be the last instance according to the lexicographical order. And let $p_{2,final}$ and p_3 be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_2^{x_{final}}$ and $\mathcal{H}_3$. Then there exist a constant $c_{2,3}$ such that $\left|p_{2,final} - p_3\right| \leq c_{2,3} \cdot \varepsilon_{iO}$.*

Lemma 5.9. *let p_3 and p_4 be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$. Then there exist a constant $c_{3,4}$ such that $|p_3 - p_4| \leq c_{3,4} \cdot \varepsilon_{iO}$.*

Lemma 5.10. *Let $\mathsf{W}^* = 0$. And let p_4 and p_{5,W^*} be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_4$ and $\mathcal{H}_4^{\mathsf{W}^*}$. Then there exist a constant $c_{4,5}$ such that $\left|p_4 - p_{5,\mathsf{W}^*}\right| \leq c_{4,5} \cdot (\varepsilon_{\mathsf{PPRF}} + \varepsilon_{iO} + \varepsilon_{\mathsf{PRG}})$.*

Lemma 5.11. *Let $\mathsf{W}^* < \bar{\mathsf{W}}$. And let p_{5,W^*} and p_{5,W^*+1} be the probabilities of obtaining 1 in the corresponding hybrids $\mathcal{H}_5^{\mathsf{W}^*}$ and $\mathcal{H}_5^{\mathsf{W}^*+1}$. Then there exist a constant c_5 such that $\left|p_{5,\mathsf{W}^*} - p_{5,\mathsf{W}^*+1}\right| \leq c_5 \cdot (\varepsilon_{\mathsf{PPRF}} + \varepsilon_{iO} + \varepsilon_{\mathsf{PRG}}) + 2^{-\lambda_{\mathsf{PRG}}}$.*

Proof of Lemma 5.5. The proof follows by the correctness of the decider $\mathsf{Dec}_{\tau(\beta)}$. □

Proof sketch of Lemmas 5.6 and 5.7. The proofs of Lemmas 5.6 and 5.7 are along the same lines, and we therefore focus on Lemma 5.7. The proof corresponds to hybrids $\mathcal{H}_{2,(\alpha,\beta,T^*),0}$ to $\mathcal{H}_{2,(\alpha,\beta,T^*),5}$, and lemmas 5.8 to 5.13 in [DJJ+25]. We sketch the sub-hybridsß:

- PM proves and Verify verifies instances $x \leq x_i$ such that $x \notin L^*_{\tau(\beta)}$ with the off path proof $\mathsf{PPRF.Eval}_{K_{\text{off}}}(x) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x))$. This hybrid distributes as $\mathcal{H}_2^{x_i}$.
- We puncture the PPRF keys K, K_{off} at the point x_{i+1}, and hardwire the value $z = \mathsf{PPRF.Eval}_K(x_{i+1})$ into the circuits PM and Verify. Indistinghuishability follows by $i\mathcal{O}$ security, since the functionality of PM and Verify is preserved.
- We replace z with a uniformly random value. Indistinghuishability follows by PPRF security at the punctured point x_{i+1}.
- We set $z = r \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_{i+1}))$, for uniformly random value r. This does not affect the distribution of z.
- We replace r with $\mathsf{PPRF.Eval}_{K_{\text{off}}}(x_{i+1}) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x_{i+1}))$. Indistinghuishability follows by PPRF security at the punctured point x_{i+1}.
- We remove the puncturing of the PPRF keys K, K_{off} and the hardwiring of the value z. Indistinghuishability follows by $i\mathcal{O}$, since the functionality of PM and Verify is preserved. This hybrid distributes as $\mathcal{H}_2^{x_{i+1}}$.

□

Proof of Lemma 5.8. Indistinghuishability follows by $i\mathcal{O}$ security, since the circuits have the same functionality. □

Proof of Lemma 5.9. By the injectivity of G, it follows that

$$\pi = \mathsf{PPRF.Eval}_{K_{\text{off}}}(x) \oplus \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x))$$

if and only if it holds that

$$G(\pi \oplus \mathsf{PPRF.Eval}_{K_{\text{off}}}(x)) = G(\mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}(x))) \ .$$

Indistinguishability therefore follows by $i\mathcal{O}$ security, since the functionality of PM and Verify is preserved □

Next we prove Lemmas 5.10 and 5.11, following the technique of [DJJ+25]. The main step is hybrid $\mathcal{H}_{5,1}^{\mathsf{W}^*}$ corresponding to hybrid $\mathcal{H}_{4,T^*,1}$ in [DJJ+25].

Proof sketch of Lemmas 5.10 and 5.11. The proofs of Lemmas 5.10 and 5.11 are along the same lines, and we therefore focus on Lemma 5.11. The proof corresponds to hybrids $\mathcal{H}_{4,T^*,0}$ to $\mathcal{H}_{4,T^*,6}$, and lemmas 5.16 to 5.21 in [DJJ+25]. We sketch the sub-hybrids:

- PM no longer provides proof for instances $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \leq \mathsf{W}^*$. Similarly, Verify no longer verifies instances $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \leq \mathsf{W}^*$. This hybrid distributes as $\mathcal{H}_5^{\mathsf{W}^*}$.

- PM no longer provides proof for instances $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) \leq \mathsf{W}^* + 1$. This hybrid corresponds to $\mathcal{H}^{\mathsf{W}^*}_{5,1}$, which corresponds to hybrid $\mathcal{H}_{4,T^*,1}$ in [DJJ+25].

  To prove indistinghuishability we show that the functionality of PM is preserved, and therefore indistinghuishability follows by $i\mathcal{O}$ security. Consider the PM circuit in $\mathcal{H}^{\mathsf{W}^*}_5$, and let $x = \mathsf{IM}(\beta, x_0, x_1)$ such that $\mathsf{W}(x) = \mathsf{W}^* + 1$. By the soundness of the instance merger IM, if $x \notin L^*_{\tau(\beta)}$ and $x \neq \perp$, then there must be $i \in \{0,1\}$ such that $x_i \notin L^*_{\tau(\beta)}$. By the monotonicity of the weight function, since $\mathsf{W}(x) = \mathsf{W}^* + 1$, it follows that $\mathsf{W}(x_i) \leq \mathsf{W}^*$. Therefore, x_i does not pass the verification step of PM, so PM will not generate a proof for x.

- We puncture the PPRF key K_{chal} at the point $\mathsf{W}^* + 1$, and let $y = \mathsf{PPRF.Eval}_{K_{\text{chal}}}(\mathsf{W}^* + 1)$. We hardwire the value $z = G(y)$ into the circuits PM and Verify. In particular, at this stage PM does not compute the value y as part of any proof generation. Indistinghuishability follows by $i\mathcal{O}$ security, since the functionality of PM and Verify is preserved.

- We replace y with a uniformly random value. Indistinghuishability follows by PPRF security at the punctured point $\mathsf{W}^* + 1$.

- We replace z with a uniformly random value. Indistinghuishability follows by PRG security.

- PM and Verify does not verify proofs of false instances $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) = \mathsf{W}^* + 1$. To prove indistinghuishability, observe that the functionality of PM and Verify is preserved if $z \notin \mathit{Support}(G)$, and therefore and in that case indistinghuishability follows by $i\mathcal{O}$ security. Moreover, since G is length-doubling, it holds that $z \notin \mathit{Support}(G)$ with probability at least $1 - 2^{-\lambda_{\mathsf{PRG}}}$.

- We remove the puncturing of K_{chal} and the value z. The functionality of PM and Verify is preserved since the they do not verify proofs for false instance $x \notin L^*_{\tau(\beta)}$ such that $\mathsf{W}(x) = \mathsf{W}^* + 1$. Therefore, indistinghuishability follows by $i\mathcal{O}$ security. This hybrid distributes as $\mathcal{H}^{\mathsf{W}^*+1}_5$

□

Finally, we prove Claim 5.4.

Proof of Claim 5.4. Observe that by construction $p_{5,\bar{\mathsf{W}}} = 0$. Next, we invoke Lemma 5.7 2^n times, and Lemma 5.11 $\bar{\mathsf{W}}$ times. The rest of the lemmas (i.e. Lemmas 5.5, 5.6 and 5.8 to 5.10) are invoked once. Therefore, there exists a constant c such that for large enough λ it holds that

$$\begin{aligned} p_0 &\leq c \cdot (2^n \cdot (\varepsilon_{\mathsf{PPRF}} + \varepsilon_{i\mathcal{O}}) + \bar{\mathsf{W}} \cdot (\varepsilon_{\mathsf{PPRF}} + \varepsilon_{i\mathcal{O}} + \varepsilon_{\mathsf{PRG}} + 2^{-\lambda_{\mathsf{PRG}}})) \\ &\leq c \cdot (2 \cdot 2^{-2\lambda} + 2 \cdot 2^{-(\lambda+n)} + \bar{\mathsf{W}} \cdot (\varepsilon_{\mathsf{PRG}} + 2^{-\lambda_{\mathsf{PRG}}})) \end{aligned}$$

Now, if the PRG is polynomially secure and $\bar{\mathsf{W}}$ is polynomially bounded, then $\bar{\mathsf{W}} \cdot (\varepsilon_{\mathsf{PRG}} + 2^{-\lambda_{\mathsf{PRG}}})$ is a negligible function. Finally, if the PRG is subexponentially secure and $\bar{\mathsf{W}} = 2^\lambda$, then $\bar{\mathsf{W}} \cdot (\varepsilon_{\mathsf{PRG}} + 2^{-\lambda_{\mathsf{PRG}}}) = 2 \cdot 2^{-\lambda}$, which is a negligible function. □

6 Application I: CCA Secure Fully Homomorphic Encryption

In this section we construct a multi-hop CCA1 secure fully homomorphic encryption based on our mergeable arguments and non-interactive zero knowledge scheme NIZK. The construction follows the [NY89]

paradigm. We analyze the scheme assuming the NIZK scheme has statistical soundness, and the mergeable proof system has universal soundness. It is possible to relax the NIZK scheme to have computational soundness if the mergeable proof system has trapdoor extraction, see Remark 6.2. Let FHE = (FHE.Setup, FHE.Enc, FHE.Dec, FHE.Eval) be a fully homomorphic encryption scheme. In the following we define trapdoor languages for the mergeable argument scheme.

6.1 Trapdoor Languages for CCA1 FHE

We define a family of trapdoor languages. Let $\mathcal{R}$ be the following NP relation:

$$\mathcal{R} = \{((\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1), (b, r_0, r_1)) \mid s.t.\ b \in \{0,1\}, r_0, r_1 \in \{0,1\}^* \\ \wedge \mathsf{ct}_0 = \mathsf{FHE.Enc}(\mathsf{pk}_0, b; r_0) \wedge \mathsf{ct}_1 = \mathsf{FHE.Enc}(\mathsf{pk}_1, b; r_1)\}$$

And let L_{double} be the corresponding language. Let $n := n(\lambda)$ be the size of instances in L_{double} when $\mathsf{pk}_0, \mathsf{pk}_1$ are in the support of $\mathsf{FHE.Setup}(1^\lambda)$. Let NIZK = (NIZK.Setup, NIZK.Prove, NIZK.Verify, NIZK.Sim) be a non-interactive zero knowledge scheme for L_{double} with statistical soundness. For $\beta = (\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$, define the language

$$L_\beta = \left\{(0, \mathsf{ct}_0, \mathsf{ct}_1) \mid \exists \pi \in \{0,1\}^* \ s.t.\ \mathsf{NIZK.Verify}(\mathsf{crs}, \pi, (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1)) = 1\right\} .$$

Let GOOD_λ be the set of $\mathsf{crs} \in \mathsf{Support}(\mathsf{NIZK.Setup}(1^\lambda, 1^n))$ where there does not exist $(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1) \notin L_{\text{double}}$ and a proof π such that $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi, (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1)) = 1$. Define the set of indices

$$\mathrm{B}_\lambda = \{(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1) \mid \mathsf{crs} \in \mathsf{GOOD}_\lambda, \exists \mathsf{sk}_0, \mathsf{sk}_1 \in \{0,1\}^* \\ s.t.\ (\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1) \in \mathsf{Support}(\mathsf{FHE.Setup}(1^\lambda))\} .$$

Let $L = \left\{L_\beta\right\}_{\beta \in \mathrm{B}}$ be a language family where $\mathrm{B} = \{\mathrm{B}_\lambda\}_{\lambda \in \mathbb{N}}$. Next, for $\tau = (\mathsf{sk}_0, \mathsf{sk}_1)$, define the language

$$L^*_\tau = \left\{(j, \mathsf{ct}_0, \mathsf{ct}_1) \mid \mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}_0) = \mathsf{FHE.Dec}_{\mathsf{sk}_1}(\mathsf{ct}_1)\right\} ,$$

and define the set of indices

$$\mathrm{T}_\lambda = \left\{((\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1)) | (\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1) \in \mathsf{Support}(\mathsf{FHE.Setup}(1^\lambda))\right\} .$$

Let $L^* = \left\{L^*_\tau\right\}_{\tau \in \mathrm{T}}$ be a language family, where $\mathrm{T} = \{\mathrm{T}_\lambda\}_{\lambda \in \mathbb{N}}$. The languages are equipped with the natural weight function:

$$\mathrm{W}(j, \mathsf{ct}_0, \mathsf{ct}_1) = j .$$

Next we define an instance merger IM with auxiliary input (Remark 3.1) as follows. Let $x_0, x_1 \in \{0,1\}^*$, and $\beta, z \in \{0,1\}^*$. The instance merger parses the instances $x_i = (j^i, \mathsf{ct}^i_0, \mathsf{ct}^i_1)$, the public parameter $\beta = (\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$, and the auxiliary input $z = g$ as the description of a gate (where we choose some universal family of gates). The instance merger IM then outputs:

$$(\max(j^0, j^1) + 1, \mathsf{FHE.Eval}_{\mathsf{pk}_0}(g, (\mathsf{ct}^0_0, \mathsf{ct}^1_0)), \mathsf{FHE.Eval}_{\mathsf{pk}_1}(g, (\mathsf{ct}^0_1, \mathsf{ct}^1_1))) .$$

Given this instance merger, we define an instance merger for circuits (Section 4.2.1) $\mathsf{IM}(\beta, C, x_1, \ldots, x_k)$, that applies the instance merger for each gate in the circuit, and outputs the final wire $(j, \mathsf{ct}_0, \mathsf{ct}_1)$.

We are now ready to construct our CCA1 secure FHE scheme.

6.2 Construction

Let $\Pi = (\Pi.\mathsf{Gen}, \Pi.\mathsf{Prove}, \Pi.\mathsf{Verify}, \Pi.\mathsf{PM})$ be a mergeable proof system with universal soundness for (L, L^*, IM) defined above. An encryption of a message $m \in \{0,1\}^*$ is done by encrypting each bit separately, i.e. $\mathsf{FHE.Enc}_{\mathsf{pk}}(m) = \mathsf{FHE.Enc}_{\mathsf{pk}}(m_1), ..., \mathsf{FHE.Enc}_{\mathsf{pk}}(m_{|m|})$. Same goes for decryption, i.e., each bit decrypted separately. We therefore focus on encrypting and decrypting a single bit. We construct a CCA1 secure FHE′ scheme as follows:

Setup′: On input security parameter 1^λ, the setup algorithm is as follows:

- Sample $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{n(\lambda)})$.[11]
- Sample $\mathsf{pk}_\Pi \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$.
- Sample $(\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.
- Set $\mathsf{pk} = (\mathsf{pk}_\Pi, \mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1))$ and $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{pk})$, and output $(\mathsf{sk}, \mathsf{pk})$.

Enc′: On input public key pk, and bit $b \in \{0,1\}$, the encryption algorithm is as follows:

- Parse $\mathsf{pk} = (\mathsf{pk}_\Pi, (\mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1)))$.
- Sample randomness $r_0, r_1 \leftarrow \{0,1\}^\lambda$.
- Compute $\mathsf{ct}_0 = \mathsf{FHE.Enc}(\mathsf{pk}_0, b; r_0)$ and $\mathsf{ct}_1 = \mathsf{FHE.Enc}(\mathsf{pk}_1, b; r_1)$.
- Generate a proof for the NIZK scheme:

$$\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1), (b, r_0, r_1)) \ .$$

- Set $\beta = (\mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1))$ and $\mathsf{ct} = (0, \mathsf{ct}_0, \mathsf{ct}_1)$.
- Generate a proof for the mergeable proof scheme Π:

$$\pi_\Pi \leftarrow \Pi.\mathsf{Prove}(\mathsf{pk}_\Pi, \beta, \mathsf{ct}, \pi_{\mathsf{NIZK}}) \ .$$

- Output (ct, π_Π).

Dec′: On input ciphertext $\mathsf{ct} \in \{0,1\}^*$, and a key $\mathsf{sk} \in \{0,1\}^*$, the decryption algorithm is as follows:

- Parse $\mathsf{sk} = (\mathsf{sk}_0, (\mathsf{pk}_\Pi, \mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1)))$.
- Parse $\mathsf{ct} = ((k, \mathsf{ct}_0, \mathsf{ct}_1), \pi_\Pi)$.
- Set $\beta = (\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$.
- If $\Pi.\mathsf{Verify}(\mathsf{pk}_\Pi, \beta, (k, \mathsf{ct}_0, \mathsf{ct}_1)) = 1$ and $k \leq 2^\lambda$:
  - Decrypt $y = \mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}_0)$.
  - Output y.
- Else: output $\perp$.

Eval′: On input a public key $\mathsf{pk} \in \{0,1\}^*$, a circuit C with k input wires, and ciphertexts $\mathsf{ct}^1, \ldots, \mathsf{ct}^k$, the evaluation algorithm is as follows:

- Parse $\mathsf{pk} = (\mathsf{pk}_\Pi, (\mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1)))$.

[11] Recall that $n := n(\lambda)$ is the size of instances in L_{double} when $\mathsf{pk}_0, \mathsf{pk}_1$ are in the support of $\mathsf{FHE.Setup}(1^\lambda)$.

- For $i \in [k]$, parse $\mathsf{ct}^i = (\widehat{\mathsf{ct}}^i, \pi^i_\Pi)$.
- Set $\beta = (\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$.
- Compute the evaluated ciphertext using the instance merger:

$$\widehat{\mathsf{ct}}' \leftarrow \mathsf{IM}(\beta, C, (\widehat{\mathsf{ct}}^1, \ldots, \widehat{\mathsf{ct}}^k)) \,.$$

- Compute the new proof by merging previous proofs:

$$\pi'_\Pi \leftarrow \mathsf{PM}(\beta, \mathsf{pk}_\Pi, C, (\widehat{\mathsf{ct}}^1, \pi^1_\Pi), \ldots, (\widehat{\mathsf{ct}}^k, \pi^k_\Pi)) \,,$$

  where we can either use native proof merging for circuits (as in Section 4.2.1), or simply compute the proof merger for each gate separately and output the final merged proof.
- Output $\mathsf{ct}' = (\widehat{\mathsf{ct}}', \pi'_\Pi)$.

6.3 Analysis

Theorem 6.1 (Correctness and Evaluation Correctness)**.** *The scheme* FHE′ *is correct, and has evaluation correctness.*

Proof. Correctness follows from the correctness of the FHE scheme and the correctness of the NIZK scheme. Evaluation correctness follows from the evaluation correctness of the FHE scheme and the merge completeness of the mergeable proof scheme Π. □

Theorem 6.2 (Efficiency)**.** *Assume* FHE *is multi-hop compact. Then: if* Π *has depth-compact merge efficiency, then* FHE′ *is also multi-hop compact. If* Π *has merge efficiency for circuits, then* FHE′ *is multi-hop semi-compact.*

Proof. Consider an evaluated ciphertext $\widehat{\mathsf{ct}}$ according to a DAG D where each node in the DAG is a circuit C (as in Definition 2.4). The proof is by induction on the DAG depth. Consider the final hop $\widehat{\mathsf{ct}} = \mathsf{Eval}'_{\mathsf{pk}}(C, \mathsf{ct}^1, \ldots, \mathsf{ct}^k)$. By construction of the FHE′, $\widehat{\mathsf{ct}} = \widehat{\mathsf{ct}}^1, ..., \widehat{\mathsf{ct}}^{|C(m)|}$. Moreover, $\widehat{\mathsf{ct}}^i = ((\widehat{\mathsf{ct}}^i_0, \widehat{\mathsf{ct}}^i_1), \pi_\Pi)$, where $\widehat{\mathsf{ct}}^i_0$ and $\widehat{\mathsf{ct}}^i_1$ are ciphers of the FHE scheme, and π_Π is a proof for the mergeable proof system Π.

Now, by induction, if Π is compact, then there exists a polynomial p such that $|\pi_\Pi| \leq p(\lambda)$. Similarly, if Π has merge efficiency for circuits, then there exists a polynomial q such that $|\pi_\Pi| \leq q(\lambda, \mathsf{Depth}(D))$. The theorem then follows since $\widehat{\mathsf{ct}}^i_0$ and $\widehat{\mathsf{ct}}^i_1$ are evaluated ciphers of the FHE scheme, encrypting the corresponding output wire of C, hence are of size $p'(\lambda)$ if FHE is multi-hop compact, for some polynomial p'. □

Remark 6.1. We note that if Π only has standard merge efficiency, then we get a weaker multi-hop compactness guarantee where the ciphertext grows with the depth of the total computed circuit (accounting for the depth of all circuits in the DAG).

Theorem 6.3. *Assume* FHE *has semantic security,* Π *has universal soundness and* NIZK *has statistical soundness, then* FHE′ *has CCA1 security.*

Proof. Since the scheme has a public key, it is enough to show an encryption of 0 is indistinguishable from an encryption of 1. Also, by the definition of the decryption algorithm Dec′, we assume without loss of generality each decryption query made by the adversary corresponds to an encryption of a single bit. In the following hybrids, let p_i be the probability that the adversary outputs 1 in the i-th hybrid. The proof is through the following hybrids:

$\mathcal{H}_0$: This is the original security game, where $\mathsf{ct}_0 = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(0)$ and $\mathsf{ct}_1 = \mathsf{FHE.Enc}_{\mathsf{pk}_1}(0)$, and $\mathsf{ct} = ((0, \mathsf{ct}_0, \mathsf{ct}_1), \pi_{\Pi})$.

$\mathcal{H}_1$: In this hybrid we switch to simulated crs and simulated proof. Formally, the crs is simulated by the zero-knowledge simulator Sim_1 by sampling $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^{n(\lambda)})$. In addition, in the challenge phase, the proof π_{NIZK} is computed by

$$\pi_{\mathsf{NIZK}} \leftarrow \mathsf{Sim}_2(\mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1), \mathsf{td}) \ .$$

$\mathcal{H}_2$: In this hybrid we switch the challenge ciphertext ct_1 to be $\mathsf{ct}_1 = \mathsf{FHE.Enc}_{\mathsf{pk}_1}(1)$.

$\mathcal{H}_3$: In hybrid the secret key contains sk_1 instead of sk_0, i.e. $\mathsf{sk} = (\mathsf{sk}_1, (\mathsf{pk}_\Pi, \mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1)))$. Moreover, given a ciphertext $\mathsf{ct} = ((k, \mathsf{ct}'_0, \mathsf{ct}'_1), \pi_\Pi)$, the decryption oracle returns $\mathsf{FHE.Dec}_{\mathsf{sk}_1}(\mathsf{ct}'_1)$ after verifying that $\Pi.\mathsf{Verify}(\mathsf{pk}_\Pi, \beta, (k, \mathsf{ct}'_0, \mathsf{ct}'_1)) = 1$ and that $k \leq 2^\lambda$.

$\mathcal{H}_4$: In this hybrid we switch the challenge ciphertext ct_0 to be $\mathsf{ct}_0 = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(1)$.

$\mathcal{H}_5$: In this hybrid we switch back the crs to be computed by $\mathsf{NIZK.Setup}$. Moreover, the π_{NIZK} is computed by running

$$\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1), (1, r_0, r_1)) \ ,$$

where r_0, r_1 are the random coins used by the encryption algorithm.

$\mathcal{H}_6$: We switch back the secret key to contain sk_0, i.e. $\mathsf{sk} = (\mathsf{sk}_0, (\mathsf{pk}_\Pi, \mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1)))$. Moreover, given a ciphertext $\mathsf{ct} = ((k, \mathsf{ct}'_0, \mathsf{ct}'_1), \pi_\Pi)$, the decryption oracle returns $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}'_0)$ after verifying that $\Pi.\mathsf{Verify}(\mathsf{pk}_\Pi, \beta, (k, \mathsf{ct}'_0, \mathsf{ct}'_1)) = 1$ and $k \leq 2^\lambda$. Note $\mathcal{H}_7$ corresponds to the original security game when $b = 1$ is encrypted.

By combining Claims 6.4 to 6.9, and by the triangle inequality, we conclude there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ we have that $|p_0 - p_6| \leq \mu(\lambda)$, which proves CCA1 security. □

Claim 6.4. *There exists a negligible function μ_1 such that for all $\lambda \in \mathbb{N}$ we have that $|p_0 - p_1| \leq \mu_1(\lambda)$.*

Proof. This is by a direct reduction to the zero-knowledge of the NIZK scheme. We describe a distinguisher between a simulated crs and simulated proof, and a crs generated by $\mathsf{NIZK.Setup}$ and a real proof of the prover. The reduction $\mathcal{B}$ gets crs as input from its challenger. It samples keys $(\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{FHE.Setup}(\lambda)$ and $\mathsf{pk}_\Pi \leftarrow \Pi.\mathsf{Gen}(\lambda)$. It then runs $\mathcal{A}$ with $\mathsf{pk} = (\mathsf{pk}_\Pi, \mathsf{crs}, (\mathsf{pk}_0, \mathsf{pk}_1))$ and answers its decryption queries like $\mathsf{Dec}'_{\mathsf{sk}}(\cdot)$. In the challenge phase $\mathcal{B}$ sample random coins r_0, r_1 and computes $\mathsf{ct}_0 = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(0; r_0)$ and $\mathsf{ct}_1 = \mathsf{FHE.Enc}_{\mathsf{pk}_1}(0; r_1)$. It then asks its challenger to generate the π_{NIZK} proof for the statement $x = (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1)$ with the witness $w = (0, r_0, r_1)$. It then gets b' from $\mathcal{A}$ and answer 1 if $b = b'$. It is easy to see that when $\mathcal{B}$ gets a crs generated by $\mathsf{NIZK.Setup}$ and gets a proof from a real prover it perfectly simulates $\mathcal{H}_0$, and when $\mathcal{B}$ interacts with a NIZK simulator it perfectly simulates $\mathcal{H}_1$. □

Claim 6.5. *There exists a negligible function μ_1 such that for all $\lambda \in \mathbb{N}$ we have that $|p_1 - p_2| \leq \mu_2(\lambda)$.*

Proof. This is by a direct reduction to the semantic security of the FHE scheme. □

Claim 6.6. *There exists a negligible function μ_2 such that for all $\lambda \in \mathbb{N}$ we have that $|p_2 - p_3| \leq \mu_3(\lambda)$.*

Proof. Let $\beta = (\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$. Let A be the event that $\mathcal{A}$ sends a decryption query $\mathsf{ct} = ((k, \mathsf{ct}_0, \mathsf{ct}_1), \pi_\Pi)$ such that $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}_0) \neq \mathsf{FHE.Dec}_{\mathsf{sk}_1}(\mathsf{ct}_1)$ and $k \leq 2^\lambda$ and $\Pi.\mathsf{Verify}(\mathsf{pk}_\Pi, \beta, (k, \mathsf{ct}_0, \mathsf{ct}_1)) = 1$. It is enough to show that there exists a negligible function μ such that for any $\lambda \in \mathbb{N}$, it holds that $\Pr[A] \leq \mu(\lambda)$.

Indeed, let $\tau = (\mathsf{sk}_0, \mathsf{sk}_1)$ be the corresponding secret keys of the public keys $\mathsf{pk}_0, \mathsf{pk}_1$. If A didn't occur, then answering according to $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\cdot)$ is similar as to answer according to $\mathsf{FHE.Dec}_{\mathsf{sk}_1}(\cdot)$.

It is left to show that there exists a negligible function μ' such that for any $\lambda \in \mathbb{N}$, it holds that $\Pr[A] \leq \mu(\lambda)$. Let B be the same event, when $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(\lambda)$. By the statistical soundness of the NIZK scheme, there exists a negligible function μ' such that with probability at least $1 - \mu'(\lambda)$ it holds that $\mathsf{crs} \in \mathsf{GOOD}_\lambda$. Consider now the mergeable proof system Π, and consider the (inefficient) mapping $\tau : \mathrm{B}_\lambda \to \mathrm{T}_\lambda$ that maps $(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$ to $((\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1))$. This mapping is sound for IM by the evaluation correctness of the FHE scheme. Moreover, it is sound for L by the correctness of the FHE scheme and since $\mathsf{crs} \in \mathsf{GOOD}_\lambda$.

Therefore, by the universal soundness of Π, there exists a negligible function μ'' such that it holds that $\Pr\left[\ B \ : \ \mathsf{crs} \in \mathsf{GOOD}_\lambda\ \right] \leq \mu''(\lambda)$. It follows that $\Pr[B] \leq \mu'(\lambda) + \mu''(\lambda)$. Finally, by a reduction to the zero knowledge of the NIZK scheme as in Claim 6.4, it follows that there exists a negligible function μ''' such that for every $\lambda \in \mathbb{N}$ it holds that $\Pr[A] \leq \Pr[B] + \mu'''(\lambda)$.

Finally, set $\mu = \mu' + \mu'' + \mu'''$. □

Claim 6.7. *There exists a negligible function μ_3 such that for all $\lambda \in \mathbb{N}$ we have that $|p_3 - p_4| \leq \mu_4(\lambda)$.*

Proof. This is by a direct reduction to the semantic security of the FHE scheme. □

Claim 6.8. *There exists a negligible function μ_4 such that for all $\lambda \in \mathbb{N}$ we have that $|p_4 - p_5| \leq \mu_4(\lambda)$.*

Proof. This is done along the same lines as in Claim 6.4. □

Claim 6.9. *There exists a negligible function μ_5 such that for all $\lambda \in \mathbb{N}$ we have that $|p_5 - p_6| \leq \mu_6(\lambda)$.*

Proof. This is done along the same lines as in Claim 6.6. □

Remark 6.2 (Relaxing NIZK Security)**.** We note that we can relax the requirement of statistical soundness for the NIZK scheme if the mergeable proof system has universal trapdoor extraction. Indeed, the only place it is used is in Claims 6.6 and 6.9. If the NIZK scheme has only computational soundness, then the mapping $\tau : \mathrm{B}_\lambda \to \mathrm{T}_\lambda$ is not sound for L. However, by trapdoor extraction of Π, an adversary $\mathcal{A}$ that sends a decryption query $\mathsf{ct} = ((k, \mathsf{ct}_0, \mathsf{ct}_1), \pi_\Pi)$ such that $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}_0) \neq \mathsf{FHE.Dec}_{\mathsf{sk}_1}(\mathsf{ct}_1)$ and $\Pi.\mathsf{Verify}(\mathsf{pk}_\Pi, \beta, (k, \mathsf{ct}_0, \mathsf{ct}_1)) = 1$, can be turned into an adversary against the NIZK computational soundness game.

7 Application II: Adaptive Multi-Hop Aggregate Signatures

In this section we extend multi-hop aggregate signatures, first constructed from multi-hop BARGs in [DGKV22], to satisfy a stronger aggregate unforgeability guarantee that holds against adaptively chosen aggregation structures. We additionally obtain a short CRS that does not grow with the number of hops.

7.1 Definition

We define multi-hop aggregate signatures with adaptive unforgeability.

Syntax. A multi-hop aggregate signature scheme for signature scheme $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ consists of the following polynomial-time algorithms:

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

$\mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_i\right\}_{i\in[k]}) \to \widehat{\mathsf{vk}, \mathsf{m}}$. This is a deterministic algorithm that takes as input a common reference string crs, and a sequence of aggregated verification keys and messages. It outputs an aggregated verification key and message $\widehat{\mathsf{vk}, \mathsf{m}}$.

$\mathsf{SigAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_i, \hat{\sigma}_i\right\}_{i\in[k]}) \to \hat{\sigma}$. This is a deterministic algorithm that takes as input a common reference string crs, and a sequence of aggregated verification keys and messages, and aggregated signatures. It outputs an aggregated signature $\hat{\sigma}$.

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

Definition 7.1. *An adaptive multi-hop aggregate signature scheme* $(\mathsf{Setup}, \mathsf{KeyMsgAgg}, \mathsf{SigAgg}, \mathsf{AggVerify})$ *for signature scheme* $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is required to satisfy the following properties:*

Correctness. *For any $\lambda \in \mathbb{N}$, any verification key* vk*, message* $\mathsf{m} \in \{0,1\}^\lambda$ *and signature σ, it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{vk}, \mathsf{m}, \sigma) = 1 \\ \wedge\ \mathsf{AggVerify}(\mathsf{crs}, \mathsf{vk}, \mathsf{m}, \sigma) = 0\end{array} \;:\; \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\right] = 0\,.$$

Furthermore, for any set of aggregated verification keys and messages and aggregated signatures $\left\{\widehat{\mathsf{vk}, \mathsf{m}}_i, \hat{\sigma}_i\right\}_{i\in[k]}$*, it holds that*

$$\Pr\left[\begin{array}{l}\forall i \in [k].\ \mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}_i, \hat{\sigma}_i) = 1 \\ \wedge\ \mathsf{AggVerify}(\mathsf{crs}, f, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) = 0\end{array} \;:\; \begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \widehat{\mathsf{vk}, \mathsf{m}} \leftarrow \mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_i\right\}_{i\in[k]}) \\ \hat{\sigma} \leftarrow \mathsf{SigAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_i, \hat{\sigma}_i\right\}_{i\in[k]})\end{array}\right] = 0\,.$$

Efficiency. *There exists a fixed polynomial* poly *such that in the correctness experiment above* $\left|\widehat{\mathsf{vk}, \mathsf{m}}\right| = \mathsf{poly}(\lambda)$*, and* $|\hat{\sigma}| = \max_{i\in[k]}|\hat{\sigma}_i| + \mathsf{poly}(\lambda)$.

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

$$\Pr\left[\mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) = 1 \;:\; \begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{sk}^*, \mathsf{vk}^*) \leftarrow \mathsf{S.KeyGen}(1^\lambda) \\ (\widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}(\mathsf{crs}, \mathsf{vk}^*)\end{array}\right] \leq \mathsf{negl}(\lambda)\,,$$

and $\mathcal{A}$ is admissible if ρ is a valid opening of $\widehat{\mathsf{vk}, \mathsf{m}}$ *to* $(\mathsf{vk}^*, \mathsf{m}^*)$ *such that $\mathcal{A}$ did not query* m^* *to the signing oracle. We say that ρ is a valid opening if it passes the following verification algorithm:*

- *Parse* $\rho = \left\{\widehat{\mathsf{vk}, \mathsf{m}}_{\ell,i}\right\}_{0\le\ell<d, i\in[k_\ell]}$.
- *For* $\ell = 0$, *check that* $\exists i \in [k_0]$ *such that* $(\mathsf{vk}^*, \mathsf{m}^*) = \widehat{\mathsf{vk}, \mathsf{m}}_{0,i}$.
- *For* $0 < \ell < d$, *check that* $\exists i \in [k_\ell]$ *such that* $\widehat{\mathsf{vk}, \mathsf{m}}_{\ell,i} = \mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_{\ell-1,j}\right\}_{j\in[k_{\ell-1}]})$.
- *Check that* $\widehat{\mathsf{vk}, \mathsf{m}} = \mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_{d-1,j}\right\}_{j\in[k_{d-1}]})$.

7.2 Construction

We construct an adaptive multi-hop aggregate signature scheme (Setup, KeyMsgAgg, SigAgg, AggVerify), based on the following building blocks:

- A digital signature scheme (Definition 2.12)

$$\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}) .$$

- A fully-homomorphic encryption scheme (Definition 2.4)

$$\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval}) .$$

- A pseudorandom function PRF.
- A mergeable argument with universal trapdoor extraction (Definition 3.6)

$$\Pi = (\Pi.\mathsf{Gen}, \Pi.\mathsf{Prove}, \Pi.\mathsf{Verify}, \Pi.\mathsf{PM}, \Pi.\mathsf{TGen}, \Pi.\mathsf{Extract})$$

  for the language families L, L^* and instance merger IM defined below.

Aggregate signature scheme. We now describe the aggregate signature algorithms.

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

- Generate FHE keys $(\mathsf{FHE.pk}, \mathsf{FHE.sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.
- Encrypt $\mathsf{ct}^* = \mathsf{FHE.Enc}(\mathsf{FHE.pk}, 0)$.
- Generate a mergeable argument key $\Pi.\mathsf{pk} \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$.
- Output $\mathsf{crs} = (\mathsf{FHE.pk}, \mathsf{ct}^*, \Pi.\mathsf{pk})$.

$\mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_i\right\}_{i\in[k]})$ does the following:

- Parse $\mathsf{crs} = (\mathsf{FHE.pk}, \mathsf{ct}^*, \Pi.\mathsf{pk})$.
- For $i \in [k]$, if $\widehat{\mathsf{vk}, \mathsf{m}}_i = (\mathsf{vk}_i, \mathsf{m}_i)$ is a verification key and message for the base signature scheme, compute $\widehat{\mathsf{vk}, \mathsf{m}}_i = (0, \mathsf{ct}_i)$, for $\mathsf{ct}_i = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, f_{\mathsf{vk}_i,\mathsf{m}_i}, \mathsf{ct}^*)$ where $f_{\mathsf{vk}_i,\mathsf{m}_i}(z)$ does the following:
  - Parse $z = (\mathsf{vk}^*, \mathsf{k}, q)$.
  - Output 1 if $\mathsf{vk} = \mathsf{vk}^*$ and $\mathsf{PRF}(\mathsf{k}, \mathsf{m}) \equiv 0 \mod q$, and 0 otherwise.

- For $i \in [k]$, parse $\widehat{\mathsf{vk}, \mathsf{m}}_i = (W_i, \mathsf{ct}_i)$.
- Compute $\mathsf{ct} = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, f, \{\mathsf{ct}_i\}_{i\in[k]})$, where f is the logical OR function, $f(z_1, \ldots, z_k) = z_1 \vee \cdots \vee z_k$.
- Output $\widehat{\mathsf{vk}, \mathsf{m}} = (1 + \max_i W_i, \mathsf{ct})$.

$\mathsf{SigAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_i, \hat{\sigma}_i\right\}_{i\in[k]})$ does the following:

- Parse $\mathsf{crs} = (\mathsf{FHE.pk}, \mathsf{ct}^*, \Pi.\mathsf{pk})$ and let $\beta = (\mathsf{FHE.pk}, \mathsf{ct}^*)$.
- For $i \in [k]$, if $(\widehat{\mathsf{vk}, \mathsf{m}}_i, \hat{\sigma}_i) = (\mathsf{vk}_i, \mathsf{m}_i, \sigma_i)$ are for the base signature scheme, compute $\hat{\sigma}_i = \Pi.\mathsf{Prove}(\Pi.\mathsf{pk}, \beta, (0, \mathsf{vk}_i, \mathsf{m}_i), \sigma_i)$.
- For $i \in [k]$, set $x_i = \widehat{\mathsf{vk}, \mathsf{m}}_i$ and $\pi_i = \hat{\sigma}_i$.
- Output $\hat{\sigma} = \Pi.\mathsf{PM}(\Pi.\mathsf{pk}, \beta, (x_i, \pi_i)_{i\in[k]})$.

$\mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma})$ does the following:

- Parse $\mathsf{crs} = (\mathsf{FHE.pk}, \mathsf{ct}^*, \Pi.\mathsf{pk})$ and let $\beta = (\mathsf{FHE.pk}, \mathsf{ct}^*)$.
- Set $x = \widehat{\mathsf{vk}, \mathsf{m}}$ and $\pi = \hat{\sigma}$.
- Output $\Pi.\mathsf{Verify}(\Pi.\mathsf{pk}, \beta, x, \pi)$.

Language families. We define the language families $L = \left\{L_\beta\right\}_{\beta\in\mathrm{B}}$ and $L^* = \left\{L^*_\tau\right\}_{\tau\in\mathrm{T}}$, a mapping $\tau = \{\tau_\lambda : \mathrm{B}_\lambda \to \mathrm{T}_\lambda\}_{\lambda\in\mathbb{N}}$, and an instance merger IM as follows:

- The indices $\beta, \tau(\beta)$ are the image of the following sampling algorithm, which receives as input a verification key vk^*, a PRF key k, a query bound $q \in \mathbb{N}$, and randomness r_1, r_2:
  - Generate FHE keys $(\mathsf{FHE.pk}, \mathsf{FHE.sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda; r_1)$.
  - Encrypt $\mathsf{ct}^* = \mathsf{FHE.Enc}(\mathsf{FHE.pk}, (\mathsf{vk}^*, \mathsf{k}, q); r_2)$.
  - Output $\beta = (\mathsf{FHE.pk}, \mathsf{ct}^*)$, $\tau = (\mathsf{FHE.sk}, \mathsf{ct}^*)$.
- The language family $L \in \mathsf{NP}$ is given by $\mathsf{V}_L(\beta, x, w)$ as follows:
  - Parse $\beta = (\mathsf{FHE.pk}, \mathsf{ct}^*)$.
  - Parse $x = (0, \mathsf{vk}, \mathsf{m})$ and $w = \sigma$.
  - Output $\mathsf{S.Verify}(\mathsf{vk}, \mathsf{m}, \sigma)$.
- The language family $L^* \in P$ is given by $\mathsf{D}_L(\tau, x)$ as follows:
  - Parse $\tau = (\mathsf{FHE.sk}, \mathsf{ct}^*)$, and decrypt $(\mathsf{vk}^*, \mathsf{k}, q) = \mathsf{FHE.Dec}(\mathsf{FHE.sk}, \mathsf{ct}^*)$.
  - If $x = (0, \mathsf{vk}, \mathsf{m})$, output 1 if $\mathsf{vk} \neq \mathsf{vk}^*$ or $\mathsf{PRF}(\mathsf{k}, \mathsf{m}) \not\equiv 0 \bmod q$, else 0.
  - If $x = (W, \mathsf{ct})$, output 1 if $\mathsf{FHE.Dec}(\mathsf{FHE.sk}, \mathsf{ct}) \neq 0$, and 0 otherwise.
- The instance merger $\mathsf{IM}(\beta, (x_1, \ldots, x_k))$ is as follows:
  - Parse $\beta = (\mathsf{FHE.pk}, \mathsf{ct}^*)$. Let $\mathsf{crs} = (\mathsf{FHE.pk}, \mathsf{ct}^*, \perp)$.
  - Output $x = \mathsf{KeyMsgAgg}(\mathsf{crs}, \{x_i\}_{i\in[k]})$.

The weight of an instance (W, ct) is simply W. The instance merger respects the weight by definition.

7.3 Analysis

Theorem 7.2. *The scheme given in Section 7.2 is an aggregate signature scheme satisfying Definition 7.1.*

Proof of Theorem 7.2.

Correctness. We observe that by FHE evaluation correctness, it follows that τ is sound for IM. Thus, correctness follows by completeness of the mergeable argument Π.

Efficiency. Follows by efficiency of the underlying building blocks.

Adaptive aggregate unforgeability. Fix any poly-size adversary $\mathcal{A}$. Assume towards contradiction that $\mathcal{A}$ is admissible, and there exists a polynomial poly such that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) = 1 \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{sk}^*, \mathsf{vk}^*) \leftarrow \mathsf{S.KeyGen}(1^\lambda) \\ (\widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}(\mathsf{crs}, \mathsf{vk}^*) \end{array}\right] > \frac{1}{\mathsf{poly}(\lambda)}\,.$$

Let q be a bound on the number of signing queries made by $\mathcal{A}$ rounded up to the next power of 2, and denote its queries by $\mathsf{m}_1, \ldots, \mathsf{m}_q$. First, we argue that

$$\Pr\left[\begin{array}{l} \mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) = 1 \\ \wedge\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}^*) \equiv 0 \bmod q \\ \wedge\ \forall i \in [q].\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}_i) \not\equiv 0 \bmod q \end{array} \;:\; \begin{array}{l} \mathsf{k} \leftarrow \{0,1\}^\lambda \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{sk}^*, \mathsf{vk}^*) \leftarrow \mathsf{S.KeyGen}(1^\lambda) \\ (\widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}(\mathsf{crs}, \mathsf{vk}^*) \end{array}\right] > \frac{1}{\mathsf{poly}(\lambda)} \cdot \frac{1}{q} \cdot \left(1 - \frac{1}{q}\right)^q\,.$$

Indeed, this is true for a random function with range $[2^\lambda]$, and thus also for a PRF with this range. The above quantity is bounded below by a new inverse polynomial which we denote $\frac{1}{\mathsf{poly}'}$.

We now consider an alternative setup algorithm $\mathsf{Setup}^*(1^\lambda)$, defined as follows:

- Sample a PRF key $\mathsf{k} \leftarrow \{0,1\}^\lambda$.
- Sample $(\mathsf{sk}^*, \mathsf{vk}^*) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$.
- Generate language parameters (β, τ) given by the parameter sampling algorithm on input $(\mathsf{vk}^*, \mathsf{k}, q)$, and parse $\beta = (\mathsf{FHE.pk}, \mathsf{ct}^*)$.
- Generate a mergeable argument public key and trapdoor $(\Pi.\mathsf{pk}, \Pi.\mathsf{td}) \leftarrow \Pi.\mathsf{TGen}(1^\lambda, \tau)$.
- Output $\mathsf{crs} = (\mathsf{FHE.pk}, \mathsf{ct}^*, \Pi.\mathsf{pk}), \Pi.\mathsf{td}, \mathsf{vk}^*, \mathsf{k}$.

By semantic security of the FHE scheme, and trapdoor hiding of the mergeable argument system Π, we have that $\mathcal{A}$ is admissible, and

$$\Pr\left[\begin{array}{l} \mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) = 1 \\ \wedge\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}^*) \equiv 0 \bmod q \\ \wedge\ \forall i \in [q].\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}_i) \not\equiv 0 \bmod q \end{array} \;:\; \begin{array}{l} (\mathsf{crs}, \Pi.\mathsf{td}, \mathsf{vk}^*, \mathsf{k}) \leftarrow \mathsf{Setup}^*(1^\lambda) \\ (\widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}(\mathsf{crs}, \mathsf{vk}^*) \end{array}\right] > \frac{1}{\mathsf{poly}'(\lambda)} - \mathsf{negl}(\lambda)\,. \quad (1)$$

We now make the following claim.

Claim 7.3. *In the above experiment, assuming $\mathcal{A}$ is admissible and $\mathsf{PRF}(\mathsf{k}, \mathsf{m}^*) \equiv 0 \bmod q$, it holds that $\widehat{\mathsf{vk}, \mathsf{m}} \notin L^*_\tau$.*

Proof of Claim 7.3. We first observe the following converse soundness condition for the instance merger IM: for $x_1, \ldots, x_k$, it holds that if $\exists i \in [k].x_i \notin L^*_\tau$, then $x = \mathsf{IM}(\beta, x_1, \ldots, x_k) \notin L^*_\tau$. Indeed, if $x_i \notin L^*_\tau$ then:

- If $x_i = (0, \mathsf{vk}_i, \mathsf{m}_i)$, since $\mathsf{D}_L(\tau, x_i) = 0$ we have that $\mathsf{vk} = \mathsf{vk}^*$ and $\mathsf{PRF}(\mathsf{k}, \mathsf{m}) \equiv 0 \bmod q$. Therefore the output of KeyMsgAgg will decrypt to 1.
- If $x_i = (W, \mathsf{ct}_i)$, since $\mathsf{D}_L(\tau, x_i) = 0$ we have that $\mathsf{FHE.Dec}(\mathsf{FHE.sk}, \mathsf{ct}_i) \neq 0$, and thus $\widehat{\mathsf{vk}, \mathsf{m}}$ will also be non-zero (since it is the logical or).

Since $\mathcal{A}$ is admissible, ρ is a valid opening of $\widehat{\mathsf{vk}, \mathsf{m}}$ to $(\mathsf{vk}^*, \mathsf{m}^*)$, and therefore:

- Denote $\rho = \left\{\widehat{\mathsf{vk}, \mathsf{m}}_{\ell,i}\right\}_{0 \le \ell < d, i \in [k_\ell]}$.
- For $\ell = 0$, we have that $\exists i \in [k_0]$ such that $(\mathsf{vk}^*, \mathsf{m}^*) = \widehat{\mathsf{vk}, \mathsf{m}}_{0,i}$. Observe that since $\mathsf{PRF}(\mathsf{k}, \mathsf{m}^*) \equiv 0 \bmod q$, for $x^0 = \widehat{\mathsf{vk}, \mathsf{m}}_{0,i}$ we have $x^0 \notin L^*_\tau$.
- For $0 < \ell < d$, assume by induction that there exists $i \in [k_{\ell-1}]$ such that $x^{\ell-1} = \widehat{\mathsf{vk}, \mathsf{m}}_{\ell-1,i} \notin L^*_\tau$. Since ρ is valid, we have an $i \in [k_\ell]$ such that $\widehat{\mathsf{vk}, \mathsf{m}}_{\ell,i} = \mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_{\ell-1,j}\right\}_{j \in [k_{\ell-1}]})$. Since IM computes KeyMsgAgg, by the converse soundness condition we get $x^\ell = \mathsf{IM}(\beta_L, \left\{x_j\right\}_{j \in [k_{\ell-1}]}) \notin L^*_\tau$.
- Since $\widehat{\mathsf{vk}, \mathsf{m}} = \mathsf{KeyMsgAgg}(\mathsf{crs}, \left\{\widehat{\mathsf{vk}, \mathsf{m}}_{d-1,j}\right\}_{j \in [k_{d-1}]})$, by the converse condition and the inductive claim for $\ell = d - 1$, we have that $x^d = \widehat{\mathsf{vk}, \mathsf{m}} \notin L^*_\tau$.

This concludes the proof of Claim 7.3.

Using Claim 7.3 and Eq. (1), we get that

$$\Pr\left[\begin{array}{l} \mathsf{AggVerify}(\mathsf{crs}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) = 1 \\ \wedge\ \forall i \in [q].\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}_i) \not\equiv 0 \bmod q \\ \wedge\ \widehat{\mathsf{vk}, \mathsf{m}} \notin L^*_\tau \end{array} : \begin{array}{l} (\mathsf{crs}, \Pi.\mathsf{td}, \mathsf{vk}^*, \mathsf{k}) \leftarrow \mathsf{Setup}^*(1^\lambda) \\ (\widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}(\mathsf{crs}, \mathsf{vk}^*) \end{array}\right] > \frac{1}{\mathsf{poly}'(\lambda)} - \mathsf{negl}(\lambda)\,.$$

We now apply (universal) trapdoor extraction of the underlying mergeable argument Π, and obtain

$$\Pr\left[\begin{array}{l} \mathsf{S.Verify}(\mathsf{vk}, \mathsf{m}, \sigma) = 1 \\ \wedge\ \forall i \in [q].\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}_i) \not\equiv 0 \bmod q \\ \wedge\ (\mathsf{vk}, \mathsf{m}) \notin L^*_\tau \end{array} : \begin{array}{l} (\mathsf{crs}, \Pi.\mathsf{td}, \mathsf{vk}^*, \mathsf{k}) \leftarrow \mathsf{Setup}^*(1^\lambda) \\ (\widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}(\mathsf{crs}, \mathsf{vk}^*) \\ ((\mathsf{vk}, \mathsf{m}), \sigma) \leftarrow \Pi.\mathsf{Extract}(\Pi.\mathsf{td}, \widehat{\mathsf{vk}, \mathsf{m}}, \hat{\sigma}) \end{array}\right] > \frac{1}{\mathsf{poly}'(\lambda)} - \mathsf{negl}(\lambda)\,.$$

Since $(\mathsf{vk}, \mathsf{m}) \notin L^*_\tau$, it follows by construction of $\mathsf{D}_L(\tau, \cdot)$ that $\mathsf{vk} = \mathsf{vk}^*$ and $\mathsf{PRF}(\mathsf{k}, \mathsf{m}) \equiv 0 \bmod q$. Since $\forall i \in [q].\ \mathsf{PRF}(\mathsf{k}, \mathsf{m}_i) \not\equiv 0 \bmod q$, this implies that the extracted m was not queried by $\mathcal{A}$ to the signing oracle.

Therefore we can use $\mathcal{A}$ to construct an adversary against the underlying signature scheme, contradicting its unforgeability. This concludes the proof of Theorem 7.2.

Acknowledgements

Tomer Solomon is supported in part by AFOSR Award FA9550-23-1-0312, ISF Award 2338/23 and ERC Advanced Grant KolmoCrypt - 101142322. Zvika Brakerski is supported by the Horizon Europe Research and Innovation Program via ERC Project ACQUA (Grant 101087742). Omer Paneth is a member of the CheckPoint Institute of Information Security and is supported by Len Blavatnik and the Blavatnik Foundation and by AFOSR Award FA9550-23-1-0312. 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, the AFOSR, the European Union or the European Research Council Executive Agency.

A Universal SNARG for Mergeable Trapdoor Languages

In this section, we construct the argument system defined in Definition 4.7 from functional BARGs and FHE, and thus prove Theorem 4.8. The construction is based on the single-hop construction of CCA FHE in [YYS25], with the following key differences:

- We support any mergeable trapdoor language (rather than the specific language for CCA FHE). This change is mostly syntactic.
- Our argument system is *universal*: soundness holds for any circuit (of size up to 2^λ), rather than only for circuits of bounded polynomial size. This requires us to use stronger forms of their building blocks, namely:
  - We use functional BARGs rather than standard somewhere-extractable BARGs.
  - We use predicate extractable commitments with a new security property which we call *prefix consistency*.

  These strong building blocks enable us to construct a more efficient reduction, that does not suffer a polynomial factor in the circuit size (or even its depth).

A.1 Predicate Extractable Commitments with Prefix Consistency

We recall the notion of a predicate extractable commitment from [YYS25] (with some parts copied verbatim). This primitive allows one to commit a vector of values into a compact commitment and open the commitment at any position. In addition, it checks if any of the values satisfy a predicate and enables one to extract its index from the commitment given a trapdoor.

We strengthen the extractability requirement, to guarantee we extract the *first* index j^* satisfying the predicate. We add a prefix consistency guarantee, which says that it is hard to open at any index $j < j^*$ to a value satisfying the predicate. This change turns out to be what we need to achieve universality.

We also make a few additional minor modifications to the definition:

- Following [PP26], we also extend the notion of extractability to *strong* extractability, that holds even when the adversary is given the trapdoor.
- We omit the standard binding requirement, since it is not required for our application.

Syntax. Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a collection of predicates, where each $f \in \mathcal{F}_\lambda$ maps $f\colon \{0,1\}^m \to \{0,1\}$. A predicate extractable commitment PEC for the predicate family $\mathcal{F}$ consists of the following polynomial-time algorithms:

$\mathsf{Gen}(1^\lambda, f) \to (\mathsf{pk}, \mathsf{td})$. This is a probabilistic setup algorithm that takes as input a security parameter 1^λ in unary, and a predicate $f \in \mathcal{F}_\lambda$. It outputs a public key pk and a trapdoor td.

$\mathsf{Commit}(\mathsf{pk}, N, \vec{x}) \to \mathsf{com}$. This is a deterministic algorithm that takes as input a public key pk, the vector length N, and a vector $\vec{x} \in (\{0,1\}^m)^N$. It outputs a commitment com.

$\mathsf{Open}(\mathsf{pk}, N, \vec{x}, j) \to \rho$. This is a deterministic algorithm that takes as input a public key pk, the vector length N, a vector $\vec{x} \in (\{0,1\}^m)^N$, and an index $j \in [N]$. It outputs an opening ρ.

$\mathsf{Verify}(\mathsf{pk}, N, \mathsf{com}, j, x, \rho) \to 0/1$. This is a deterministic algorithm that takes as input a public key pk, the vector length N, a commitment com, an index $j \in [N]$, a message $x \in \{0,1\}^m$, and an opening ρ. It outputs 1 (accept) or 0 (reject).

$\mathsf{Extract}(\mathsf{td}, N, \mathsf{com}) \to j^*$. This is a deterministic algorithm that takes as input a trapdoor td, the vector length N, and a commitment com. It outputs an index $j^* \in [N+1]$.

Definition A.1 (Predicate Extractable Commitment with Prefix Consistency)**.** *A predicate extractable commitment* PEC *with prefix consistency for the predicate family $\mathcal{F}$ satisfies the following properties:*

Correctness. *For any $\lambda \in \mathbb{N}$, any $m, N \le 2^\lambda$, any predicate $f \in \mathcal{F}_\lambda$, any vector $\vec{x} = (x_1, \ldots, x_N) \in (\{0,1\}^m)^N$, and any index $j \in [N]$,*

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{pk}, N, \mathsf{com}, j, x_j, \rho) = 1 & : \begin{array}{l}(\mathsf{pk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, f)\\ \mathsf{com} = \mathsf{Commit}(\mathsf{pk}, N, \vec{x})\\ \rho = \mathsf{Open}(\mathsf{pk}, N, \vec{x}, j)\end{array}\end{array}\right] = 1\,.$$

Succinctness. *In the correctness experiment above, the size of the public key* pk *is* $\mathsf{poly}(\lambda, m, |f|)$*, and the size of the commitment* com *and opening ρ is* $\mathsf{poly}(\lambda, m, |f|, \log N)$*.*

Predicate hiding. *For any poly-size adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$ and $f_0, f_1 \in \mathcal{F}_\lambda$ such that $|f_0| = |f_1|$,*

$$\Pr\left[\begin{array}{ll} \mathcal{A}(\mathsf{pk}) = b & : \begin{array}{l} b \leftarrow \{0,1\}\\ (\mathsf{pk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, f_b)\end{array}\end{array}\right] \le \frac{1}{2} + \mathsf{negl}(\lambda)\,,$$

Strong extractability. *For any poly-size adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$ and predicate $f \in \mathcal{F}_\lambda$, we have*

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{pk}, N, \mathsf{com}, j, x, \rho) = 1 \\ \quad \wedge\ j = j^*\ \wedge\ f(x) = 0 & : \begin{array}{l}(\mathsf{pk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, f)\\ (N, \mathsf{com}, j, x, \rho) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{td})\\ j^* \leftarrow \mathsf{Extract}(\mathsf{td}, N, \mathsf{com})\end{array}\end{array}\right] \le \mathsf{negl}(\lambda)\,.$$

Prefix consistency. *For any poly-size adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$ and predicate $f \in \mathcal{F}_\lambda$, we have*

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{pk}, N, \mathsf{com}, j, x, \rho) = 1 \\ \quad \wedge\ j < j^*\ \wedge\ f(x) = 1 & : \begin{array}{l}(\mathsf{pk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, f)\\ (N, \mathsf{com}, j, x, \rho) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{td})\\ j^* \leftarrow \mathsf{Extract}(\mathsf{td}, N, \mathsf{com})\end{array}\end{array}\right] \le \mathsf{negl}(\lambda)\,.$$

Theorem A.2. *Assuming leveled* FHE*, there exists a predicate extractable commitment scheme satisfying Definition A.1.*

Proof of Theorem A.2.

Construction. We recall the construction from [YYS25] (we can omit the collision-resistant hash, since we do not require binding). This construction is based on the construction of predicate-extractable hash in [BBK+23], and part of what follows is copied verbatim.

$\mathsf{Gen}(1^\lambda, f)$ is defined as follows:

1. Let $(\mathsf{pk}_{\mathsf{leaf}}, \mathsf{sk}_{\mathsf{leaf}}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$, and for every $0 \le j \le \lambda$ let $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.
2. Let $c_0 \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(\mathsf{sk}_{\mathsf{leaf}})$, and for every $1 \le j \le \lambda$ let $c_j \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_j}(\mathsf{sk}_{j-1})$.
3. Let $c_f = c_{f,0} \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(f)$.
4. Output $\mathsf{pk} = (\mathsf{pk}_{\mathsf{leaf}}, \{\mathsf{pk}_i, c_i\}_{0 \le i \le \lambda}, c_f)$, $\mathsf{td} = \{\mathsf{sk}_i\}_{0 \le i \le \lambda}$.

$\mathsf{Commit}(\mathsf{pk}, N, \vec{x})$ is as follows:

1. Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{leaf}}, \{\mathsf{pk}_i, c_i\}_{0 \le i \le \lambda}, c_f)$.
2. Assume wlog that N is a power of 2, $N = 2^n$ where $n \le \lambda$.
3. We consider a binary tree of height n whose vertices are represented as $v \in \{0,1\}^j$ for all $0 \le j \le n$ (where the root is represented by the empty string ε). Compute a ciphertext for each of the vertices, as follows:
   (a) For every leaf $v \in \{0,1\}^n = [N]$, we compute $\mathsf{leaf}_v = \mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(x_v)$ using fixed all-zero randomness.
   (b) For $j = 0$, we proceed as follows:
      i. Let $c_{f,0} = c_f$.
      ii. For every vertex $v \in \{0,1\}^n = [N]$, we compute $\mathsf{ct}_v = \mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_{v,\mathsf{leaf}_v}, (c_0, c_{f,0}))$, where $g_{v,\mathsf{leaf}_v}(\mathsf{sk}_{\mathsf{leaf}}, f)$ is as follows:
         - Compute $x_v = \mathsf{FHE.Dec}_{\mathsf{sk}_{\mathsf{leaf}}}(\mathsf{leaf}_v)$.
         - If $f(x_v) = 1$, output v. Otherwise, output $\perp$.
   (c) For every $1 \le j \le n$, we proceed as follows:
      i. Compute $c_{f,j} = \mathsf{FHE.Eval}_{\mathsf{pk}_j}(g_{c_{f,j-1}, c_j})$, where $g_{c_{f,j-1}}(\mathsf{sk}_{j-1})$ outputs $f = \mathsf{FHE.Dec}_{\mathsf{sk}_{j-1}}(c_{f,j-1})$.
      ii. For every vertex $v \in \{0,1\}^{n-j}$, we compute $\mathsf{ct}_v = \mathsf{FHE.Eval}_{\mathsf{pk}_j}(g_{\mathsf{ct}_{v||0}, \mathsf{ct}_{v||1}}, (c_j, c_{f,j}))$, where $g_{\mathsf{ct}_{v||0}, \mathsf{ct}_{v||1}}(\mathsf{sk}_{j-1}, f)$ is as follows:
         - For every $i \in \{0,1\}$, compute $j_i = \mathsf{FHE.Dec}_{\mathsf{sk}_{j-1}}(\mathsf{ct}_{v||i})$.
         - If there exists an $i \in \{0,1\}$ such that $j_i \neq \perp$ and $v||i$ is a prefix of j_i (in other words, j_i is a descendant of $v||i$), take the minimum such i and output j_i.
         - Otherwise, output $\perp$.
4. Output $\mathsf{com} = \mathsf{ct}_\varepsilon$.

$\mathsf{Open}(\mathsf{pk}, N, \vec{x}, j)$ is as follows:

1. Write $j \in [N]$ as a string of bits $j = (j_1, \ldots, j_n) \in \{0,1\}^n$.

2. For every $0 \le i \le n$, let $v_i = (j_1, \ldots, j_{n-i-1}, j_{n-i})$, and $v'_i = (j_1, \ldots, j_{n-i-1}, 1 - j_{n-i})$.
   In other words, v_i is the ith node in the path from j to the root ε, and v'_i is v_i's sibling.
3. Compute the tree as in Commit$(\mathsf{pk}, N, \vec{x})$, and all leaf and ct ciphertexts.
4. Output $\rho = \left(\mathsf{leaf}_j, \left\{\mathsf{ct}_{v_i}, \mathsf{ct}_{v'_i}\right\}_{0 \le i \le n}\right)$.

Verify$(\mathsf{pk}, N, \mathsf{com}, j, x, \rho)$ is as follows:

1. Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{leaf}}, \left\{\mathsf{pk}_i, c_i\right\}_{0 \le i \le \lambda}, c_f)$.
2. Parse $\rho = \left(\mathsf{leaf}^*_j, \left\{\mathsf{ct}^*_{v_i}, \mathsf{ct}^*_{v'_i}\right\}_{0 \le i \le n}\right)$.
3. Accept if all the following checks pass:
   (a) Check that $1 \le j \le N \le 2^\lambda$.
   (b) Check that $\mathsf{leaf}^*_j = \mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(x)$ with fixed all-zero randomness.
   (c) For $i = 0$, proceed as follows:
      i. Let $c_{f,0} = c_f$.
      ii. Recall that $v_0 = j$ and check that $\mathsf{ct}^*_j = \mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_{j,\mathsf{leaf}^*_j}, (c_0, c_{f,0}))$, where g_{j,leaf^*_j} is defined as in Commit.
   (d) For every $1 \le i \le n$, proceed as follows:
      i. Let $c_{f,i} = \mathsf{FHE.Eval}_{\mathsf{pk}_i}(g_{c_{f,i-1}}, c_i)$, where $g_{c_{f,i-1}}$ is defined as in Commit.
      ii. Check that $\mathsf{ct}^*_{v_i} = \mathsf{FHE.Eval}_{\mathsf{pk}_i}(g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}, (c_i, c_{f,i}))$, where $g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}$ is defined as in Commit.
   (e) Recall that $v_n = \varepsilon$ and check that $\mathsf{com} = \mathsf{ct}^*_\varepsilon$.

Extract$(\mathsf{td}, N, \mathsf{com})$ is as follows:

1. Parse $\mathsf{td} = \left\{\mathsf{sk}_i\right\}_{0 \le i \le \lambda}$.
2. Let $N = 2^n$ where $n \le \lambda$.
3. Compute $j = \mathsf{FHE.Dec}_{\mathsf{sk}_n}(\mathsf{com})$.
4. Output j if $j \in [N]$, otherwise output $N + 1$.

Analysis. We focus on proving the prefix consistency property, since the rest follow identically to [YYS25, BBK+23]. We show that prefix consistency holds perfectly: let $\mathcal{A}$ be any adversary, let $\lambda \in \mathbb{N}$ and let $f \in \mathcal{F}_\lambda$. We argue that

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{pk}, N, \mathsf{com}, j, x, \rho) = 1 \\ \wedge\ j < j^* \ \wedge\ f(x) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, f) \\ (N, \mathsf{com}, j, x, \rho) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{td}) \\ j^* \leftarrow \mathsf{Extract}(\mathsf{td}, N, \mathsf{com}) \end{array}\right] = 0\ .$$

In the above experiment, we parse:

- $\mathsf{pk} = (\mathsf{pk}_{\mathsf{leaf}}, \left\{\mathsf{pk}_i, c_i\right\}_{0 \le i \le \lambda}, c_f)$.
- $\mathsf{td} = \left\{\mathsf{sk}_i\right\}_{0 \le i \le \lambda}$.
- $\rho = \left(\mathsf{leaf}^*_j, \left\{\mathsf{ct}^*_{v_i}, \mathsf{ct}^*_{v'_i}\right\}_{0 \le i \le n}\right)$.

Assume that $\mathsf{Verify}(\mathsf{pk}, N, \mathsf{com}, j, x, \rho) = 1$ and $f(x) = 1$. We show by induction that for every $0 \le i \le n$ we have $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = j^*_i$ where j^*_i is a descendant of v_i and $1 \le j^*_i \le j$. This suffices, since we also check that $\mathsf{ct}^*_{v_n} = \mathsf{com}$, so we must have that $\mathsf{FHE.Dec}_{\mathsf{sk}_n}(\mathsf{com}) = j^*_n$ is such that $1 \le j^*_n \le j$. Since we also check $j \le N$, this implies $\mathsf{Extract}(\mathsf{td}, N, \mathsf{com}) = j^*_n \le j$, and thus the event above cannot happen.

For the base case of $i = 0$ and $v_0 = j$, we have that $c_0, c_{f,0}$ are honest encryptions of $\mathsf{sk}_{\mathsf{leaf}}, f$, so using evaluation correctness of the FHE scheme, we get

$$j^*_0 = \mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}^*_j) = \mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_{j,\mathsf{leaf}^*_j}, (c_0, c_{f,0}))) = g_{j,\mathsf{leaf}^*_j}(\mathsf{sk}_{\mathsf{leaf}}, f).$$

Now, by definition of g_{j,leaf^*_j}, and since leaf^*_j is $\mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(x)$ and we have $f(x) = 1$, we get $j^*_0 = j$, which satisfies our condition.

Now, assume the claim holds for $i-1$. Similarly, since $c_i, c_{f,i}$ are honest encryptions of sk_{i-1}, f, evaluation correctness of the FHE scheme implies that

$$j^*_i = \mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = \mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{FHE.Eval}_{\mathsf{pk}_i}(g_{\mathsf{ct}^*_{v_{i-1}},\mathsf{ct}^*_{v'_{i-1}}}, (c_i, c_{f,i}))) = g_{\mathsf{ct}^*_{v_{i-1}},\mathsf{ct}^*_{v'_{i-1}}}(\mathsf{sk}_{i-1}, f).$$

Consider j^*_{i-1} and $j^{*'}_{i-1} = \mathsf{FHE.Dec}_{\mathsf{sk}_{i-1}}(\mathsf{ct}^*_{v'_{i-1}})$. By definition of $g_{\mathsf{ct}^*_{v_{i-1}},\mathsf{ct}^*_{v'_{i-1}}}$, and since we know by the induction hypothesis that j^*_{i-1} is a descendant of v_{i-1}, we have that j^*_i is either j^*_{i-1} or $j^{*'}_{i-1}$. We split to cases:

- If it is the former, this concludes the proof, since j^*_{i-1} is also a descendant of v_i, and satisfies $1 \le j^*_{i-1} \le j$.
- The latter case may only occur if $j^{*'}_{i-1}$ is a descendant of v'_{i-1} and $v'_{i-1} = v_i||0$ is the left son (since we choose the minimum). In this case, we have that $1 \le j^{*'}_{i-1} \le v'_{i-1} < j^*_{i-1} \le j$, where the inequality $v'_{i-1} < j^*_{i-1}$ holds since j^*_{i-1} is a descendant of the right son $v_{i-1} = v_i||1$.

This shows that the inductive claim holds for i, and finishes the proof of Theorem A.2.

A.2 Construction

We now describe the construction of our argument system. It uses the following building blocks:

- A predicate extractable commitment with prefix consistency (Definition A.1)

  $$(\mathsf{Gen}_{\mathsf{PEC}}, \mathsf{Commit}_{\mathsf{PEC}}, \mathsf{Open}_{\mathsf{PEC}}, \mathsf{Verify}_{\mathsf{PEC}}, \mathsf{Extract}_{\mathsf{PEC}})$$

  for the predicate family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where each $f_\tau \in \mathcal{F}_\lambda$ is associated with a trapdoor $\tau \in \mathrm{T}_\lambda$ and is defined by $f_\tau(x) = 1$ if $\mathrm{D}_{L^*}(\tau, x) = 0$.

- A functional BARG (Definition 2.1)

  $$(\mathsf{Gen}_{\mathsf{BARG}}, \mathsf{Prove}_{\mathsf{BARG}}, \mathsf{Verify}_{\mathsf{BARG}}, \mathsf{Extract}_{\mathsf{BARG}})$$

  for the function family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$, where each $g_{\mathsf{td}_{\mathsf{PEC}}} \in \mathcal{G}_\lambda$ is associated with a trapdoor $\mathsf{td}_{\mathsf{PEC}}$ to the predicate-extractable commitment and is defined by $g_{\mathsf{td}_{\mathsf{PEC}}}(\mathsf{com}, s) = \mathsf{Extract}_{\mathsf{PEC}}(\mathsf{td}_{\mathsf{PEC}}, s, \mathsf{com})$ if it is in $[s]$, or s otherwise.

$\mathsf{TGen}(1^\lambda, \tau)$ does the following:

- Let $f = f_\tau$, and generate $(\mathsf{pk}_{\mathsf{PEC}}, \mathsf{td}_{\mathsf{PEC}}) \leftarrow \mathsf{Gen}_{\mathsf{PEC}}(1^\lambda, f)$.
- Let $g = g_{\mathsf{td}_{\mathsf{PEC}}}$, and generate $(\mathsf{pk}_{\mathsf{BARG}}, \mathsf{td}_{\mathsf{BARG}}) \leftarrow \mathsf{Gen}_{\mathsf{BARG}}(\lambda, g)$.
- Output $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{PEC}}), \mathsf{td} = \mathsf{td}_{\mathsf{PEC}}$.

$\mathsf{Gen}(1^\lambda)$ is identical to TGen except f, g are chosen arbitrarily (but of the same size).

$\mathsf{Prove}(\beta, \mathsf{pk}, \mathsf{C}, x_1, \ldots, x_k)$ does the following:

- Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{PEC}})$.
- Compute all instances in the evaluation of the instance merger for circuits $\mathsf{IM}(\beta, \mathsf{C}, x_1, \ldots, x_k)$ and denote them by $\vec{x} = (x_1, \ldots, x_s))$, ordered in topological order such that $x = x_s$ is the output instance.
- Commit to the instances $\mathsf{com} = \mathsf{Commit}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \vec{x})$.
- For every $i \in [s]$, compute an opening $\rho_i = \mathsf{Open}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \vec{x}, i)$.
- Let $\mathcal{M}[\beta, x, s, k, \mathsf{pk}_{\mathsf{PEC}}, \mathsf{com}]$ be a machine that given i, w_i does the following:
  - If $i \leq k$ is an input wire, accept.
  - Otherwise, if $k < i \leq s$ is an intermediate wire:
    - Parse $w_i = (x_i, \rho_i, \{i_j, x_{i_j}, \rho_{i_j}\}_{j \in [m]})$. Denote $i_0 = i$.
    - For all $j \in [m]$, check that $1 \leq i_j < i$.
    - For all $0 \leq j \leq m$, check that $\mathsf{Verify}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \mathsf{com}, i_j, x_{i_j}, \rho_{i_j}) = 1$.
    - Check that $\mathsf{IM}(\beta, x_{i_1}, \ldots, x_{i_m}) = x_i$.
    - If $i = s$ is the output wire, check that $x_s = x$.
    - Accept if all checks pass.

  Let T be a bound on $\mathcal{M}$'s running time.
- For every $i \in [s]$ (representing a gate in the merge circuit), construct a witness w_i for $\mathcal{M}$, where x_i is the output instance of the instance merger on the gate inputs x_{i_j}, and ρ are the previously computed openings.
- Define the tag $z = (\mathsf{com}, s)$.
- Compute $\pi_{\mathsf{BARG}} = \mathsf{Prove}_{\mathsf{BARG}}(\mathsf{pk}_{\mathsf{BARG}}, z, (\mathcal{M}, s, 1^T), \vec{w})$.
- Output the proof $\pi = (s, \mathsf{com}, \pi_{\mathsf{BARG}})$ and openings $\rho_1, \ldots, \rho_k$.

$\mathsf{Verify}(\beta, \mathsf{pk}, x, \pi, k, j, x_j, \rho_j)$ does the following:

- Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{PEC}})$ and $\pi = (s, \mathsf{com}, \pi_{\mathsf{BARG}})$.
- Define $\mathcal{M}[\beta, x, s, k, \mathsf{pk}_{\mathsf{PEC}}, \mathsf{com}]$ as above, and let T be a bound on its running time.
- Define the tag $z = (\mathsf{com}, s)$.
- Accept if all the following checks pass:
  - Check that $k, s \in [2^\lambda]$.
  - Check that $\mathsf{Verify}_{\mathsf{BARG}}(\mathsf{pk}_{\mathsf{BARG}}, z, (\mathcal{M}, s, T), \pi_{\mathsf{BARG}}) = 1$.
  - Check that $j \in [k]$ and $\mathsf{Verify}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \mathsf{com}, j, x_j, \rho_j) = 1$.

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

- Parse $\mathsf{td} = \mathsf{td}_{\mathsf{PEC}}$ and $\pi = (s, \mathsf{com}, \pi_{\mathsf{BARG}})$.
- Extract $j = \mathsf{Extract}_{\mathsf{PEC}}(\mathsf{td}_{\mathsf{PEC}}, s, \mathsf{com})$.
- Output j, or 1 if $j \notin [k]$.

A.3 Analysis

We show the construction described in Appendix A.2 satisfies Definition 4.7, thus proving Theorem 4.8.

Completeness and efficiency follows from completeness and efficiency of the underlying primitives (the proof length for the BARG scales with the size of a single witness, which is is $m \cdot (n + \mathsf{poly}(\lambda))$. To get this efficiency, we use a BARG with good rate, e.g. rate-1). Trapdoor hiding follows from key indistinguishability of the functional BARG, together with predicate hiding of the predicate extractable commitment.

We turn to prove strong trapdoor extraction. Let τ be a mapping from L to L^* that is sound for IM. Let $\mathcal{A}$ be a poly-size adversary. We want to show that there exists a negligible function negl such that for each $\lambda \in \mathbb{N}$ and $\beta \in \mathrm{B}_\lambda$,

$$\Pr\left[\begin{array}{l} x \notin L^*_{\tau(\beta)} \ \wedge \ j = j^* \wedge \\ \mathsf{Verify}(\beta, \mathsf{pk}, x, \pi, k, j, x_j, \rho_j) = 1 \end{array} \rightarrow \ x_j \notin L^*_{\tau(\beta)} \ \middle| \ \begin{array}{l} (\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, \tau(\beta)) \\ (x, \pi, k, j, x_j, \rho_j) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{td}) \\ j^* \leftarrow \mathsf{Extract}(\mathsf{td}, \pi) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ . \tag{2}$$

We consider the experiment EXP, that is identical to the above, and where we additionally do the following:

- Let $f_\tau, g_{\mathsf{td}_{\mathsf{PEC}}}, \mathsf{td}_{\mathsf{BARG}}$ obtained when running TGen.
- Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{PEC}})$ and $\mathsf{td} = \mathsf{td}_{\mathsf{PEC}}$.
- Parse $\pi = (s, \mathsf{com}, \pi_{\mathsf{BARG}})$.
- Let $\mathcal{M}, z$ as defined in Verify.
- Let $i_{\mathsf{PEC}} = \mathsf{Extract}_{\mathsf{PEC}}(\mathsf{td}_{\mathsf{PEC}}, s, \mathsf{com})$.
- Let $i = g_{\mathsf{td}_{\mathsf{PEC}}}(z)$, where recall that $i = i_{\mathsf{PEC}}$ if $i_{\mathsf{PEC}} \in [s]$, or s otherwise.
- Extract $w = \mathsf{Extract}_{\mathsf{BARG}}(\mathsf{td}_{\mathsf{BARG}}, \pi_{\mathsf{BARG}})$.
- Parse $w = (x_i, \rho_i, \{i_\ell, x_{i_\ell}, \rho_{i_\ell}\}_{\ell \in [m]})$.

Step I: Use BARG somewhere extraction. We first construct an adversary against the BARG. For any $(\mathsf{pk}_{\mathsf{PEC}}, \mathsf{td}_{\mathsf{PEC}}) \leftarrow \mathsf{Gen}_{\mathsf{PEC}}(1^\lambda, f_\tau)$, consider the adversary $\mathcal{A}_{\mathsf{BARG}}$, which has $\beta, \mathsf{pk}_{\mathsf{PEC}}, \mathsf{td}_{\mathsf{PEC}}$ hardcoded and does the following given $\mathsf{pk}_{\mathsf{BARG}}$:

- Let $\mathsf{pk} = (\mathsf{pk}_{\mathsf{BARG}}, \mathsf{pk}_{\mathsf{PEC}})$ and $\mathsf{td} = \mathsf{td}_{\mathsf{PEC}}$.
- Compute $(x, \pi, k, j, x_j, \rho_j) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{td})$.
- Parse $\pi = (s, \mathsf{com}, \pi_{\mathsf{BARG}})$.
- Let $z = (\mathsf{com}, s)$ and $\mathcal{M} = \mathcal{M}[\beta, x, s, k, \mathsf{pk}_{\mathsf{PEC}}, \mathsf{com}]$ defined as in Verify above.

- Output $(z, \mathcal{M}, k, \pi_{\mathsf{BARG}})$.

By BARG somewhere extraction for the function $g_{\mathsf{td}_{\mathsf{PEC}}}$, we have that except with negligible probability, $(\mathcal{M}, (g_{\mathsf{td}_{\mathsf{PEC}}}(z), w), T) \in L_{\mathsf{U}}$. Thus the same holds in EXP above. That is, except with negligible probability, $\mathcal{M}(i, w) = 1$.

Step II: Split to cases based on the extracted index. We now split to cases, based on i_{PEC}. Recall that the predicate extractable commitment scheme guarantees that $i_{\mathsf{PEC}} \in [s + 1]$. Therefore, we have 3 cases:

- $1 \leq i_{\mathsf{PEC}} \leq k$, i.e. we found an input wire. In this case we will obtain $x_j \notin L^*_{\tau(\beta)}$.
- $k < i_{\mathsf{PEC}} \leq s$, i.e. we found an intermediate wire. We will show this case occurs with negligible probability.
- $i_{\mathsf{PEC}} = s + 1$, i.e. we did not find any wire. We will show this case occurs with negligible probability.

We now handle each case separately. We assume that $x \notin L^*_\tau$, $j = j^*$ and Verify accepts.

Case I: found an input wire. Suppose that $1 \leq i_{\mathsf{PEC}} \leq k$. Then we have by definition of Extract that $j = j^* = i_{\mathsf{PEC}}$. Therefore, since Verify accepts and Verify checks that $\mathsf{Verify}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \mathsf{com}, j, x_j, \rho_j) = 1$, it follows from strong extractability of the PEC scheme that $f_\tau(x_j) = 1$, which implies that $x_j \notin L^*_\tau$.

Case II: found an intermediate wire. Suppose that $k < i_{\mathsf{PEC}} \leq s$. Then we have $i = i_{\mathsf{PEC}}$ and therefore $\mathcal{M}(i_{\mathsf{PEC}}, w) = 1$.

First, since $\mathcal{M}$ checks $\mathsf{Verify}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \mathsf{com}, i, x_i, \rho_i) = 1$, by strong extractability of the PEC scheme we have that $f_\tau(x_i) = 1$, which implies that $x_i \notin L^*_\tau$.

For any $\ell \in [m]$, since $\mathcal{M}$ checks $\mathsf{Verify}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \mathsf{com}, i_\ell, x_{i_\ell}, \rho_{i_\ell}) = 1$ and $i_\ell < i$, by prefix consistency of the PEC scheme we have that $f_\tau(x_{i_\ell}) = 0$, which implies that $x_{i_\ell} \in L^*_\tau$.

Finally, since $\mathcal{M}$ checks that $\mathsf{IM}(\beta, x_{i_1}, \ldots, x_{i_m}) = x_i$, and since τ is sound for IM, we get that $x_i \in L^*_\tau$, which is a contradiction.

Case III: did not find any wire. Suppose that $i_{\mathsf{PEC}} = s + 1$. Then we have $i = s$ (by definition of $g_{\mathsf{td}_{\mathsf{PEC}}}$), and therefore $\mathcal{M}(s, w) = 1$.

Since $\mathcal{M}$ checks $\mathsf{Verify}_{\mathsf{PEC}}(\mathsf{pk}_{\mathsf{PEC}}, s, \mathsf{com}, i, x_i, \rho_i) = 1$ and $x_i = x$, and since $i < i_{\mathsf{PEC}}$, by prefix consistency of the PEC scheme we have $f_\tau(x) = 0$, which implies $x \in L^*_\tau$, in contradiction to our assumption.

This concludes the proof of Theorem 4.8.

References

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Berlin, Heidelberg, August 2004. 12

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 252–283. Springer, Cham, August 2023. 9, 57, 58

[BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part IV*, volume 14654 of *LNCS*, pages 168–195. Springer, Cham, May 2024. 8, 11

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. 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 1–18. Springer, 2020. 1

[BCPR16] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. *SIAM J. Comput.*, 45(5):1910–1952, 2016. 1

[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, Cham, December 2019. 14

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008. 5

[BP15] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 236–261. Springer, Berlin, Heidelberg, November / December 2015. 1

[BPSS23] Nir Bitansky, Omer Paneth, Dana Shamir, and Tomer Solomon. Non-interactive universal arguments. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 132–158. Springer, Cham, August 2023. 6

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 474–502. Springer, 2016. 35

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 350–366. ACM, 2012. 4

[CGKS23] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 465–489. Springer, Cham, July 2023. 3

[CGSY24] Alessandro Chiesa, Ziyi Guan, Shahar Samocha, and Eylon Yogev. Security bounds for proof-carrying data from straightline extractors. In Elette Boyle and Mohammad Mahmoody, editors, *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part II*, volume 15365 of *Lecture Notes in Computer Science*, pages 464–496. Springer, 2024. 1

[CHL+25] Suvradip Chakraborty, Dennis Hofheinz, Roman Langrehr, Jesper Buus Nielsen, Christoph Striecks, and Daniele Venturi. Malleable snarks and their applications. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 184–213. Springer, 2025. 1, 2, 4

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022. 9, 14

[CRRV17] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. Chosen-ciphertext secure fully homomorphic encryption. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 213–240. Springer, Berlin, Heidelberg, March 2017. 4, 11

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331. Tsinghua University Press, 2010. 1

[DDN03] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Rev.*, 45(4):727–784, 2003. 2

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022. 5, 6, 8, 11, 12, 49

[DJJ+25] Pratish Datta, Abhishek Jain, Zhengzhong Jin, Alexis Korb, Surya Mathialagan, and Amit Sahai. Incrementally verifiable computation for NP from standard assumptions. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VII*, volume 16006 of *Lecture Notes in Computer Science*, pages 611–642. Springer, 2025. 3, 6, 10, 33, 35, 36, 43, 44

[DMP26] Nico Döttling, Giulio Malavolta, and Omer Paneth. Batch arguments with optimal communication. Cryptology ePrint Archive, Paper 2026/003, 2026. 7, 8

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, pages 792–807, 1986. 17, 35

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *i*-hop homomorphic encryption and rerandomizable yao circuits. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2010. 4

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and ppad-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 652–673. Springer, 2020. 6

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Cham, November 2021. 9

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 33–43. ACM, 1989. 2, 11, 19, 44

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1045–1056. IEEE, 2022. 6, 8, 12

[PP26] Omer Paneth and Rafael Pass. Verifiable divide-and-conquer. Cryptology ePrint Archive, Paper 2026/662, 2026. 2, 3, 6, 8, 13, 14, 55

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. Cryptology ePrint Archive, Paper 2019/158, 2019. 18

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 475–484, New York, NY, USA, 2014. Association for Computing Machinery. 16, 17, 35

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008. 1, 6, 12

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Berlin, Heidelberg, May 2005. 12

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Cham, August 2022. 11

[YYS25] Rupeng Yang, Zuoxia Yu, and Willy Susilo. Fully homomorphic encryption with chosen-ciphertext security from LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 371–405. Springer, Cham, August 2025. 4, 9, 55, 57, 58