The Power of NAPs:
Compressing OR-Proofs via Collision-Resistant Hashing

Katharina Boudgoust[1⋆] and Mark Simkin[2⋆⋆]

[1] CNRS, Univ Montpellier, LIRMM
[2] Independent Researcher

Abstract. Proofs of partial knowledge, first considered by Cramer, Damgård and Schoenmakers (CRYPTO'94) and De Santis et al. (FOCS'94), allow for proving the validity of k out of n different statements without revealing which ones those are. In this work, we present a new approach for transforming certain proofs system into new ones that allows for proving partial knowledge. The communication complexity of the resulting proof system only depends logarithmically on the total number of statements n and its security only relies on the existence of collision-resistant hash functions. As an example, we show that our transformation is applicable to the proof systems of Goldreich, Micali, and Wigderson (FOCS'86) for the graph isomorphism and the graph 3-coloring problem.

Our main technical tool, which we believe to be of independent interest, is a new cryptographic primitive called non-adaptively programmable functions (NAPs). Those functions can be seen as pseudorandom functions which allow for re-programming the output at an input point, which must be fixed during key generation. Even when given the re-programmed key, it remains infeasible to find out where re-programming happened. Finally, as an additional technical tool, we also build explainable samplers for any distribution that can be sampled efficiently via rejection sampling and use them to construct NAPs for various output distributions.

1 Introduction

Proofs of partial knowledge, independently first considered by Cramer, Damgård and Schoenmakers [CDS94] and De Santis et al. [DDP+94], allow a prover to convince a verifier that k out of a list of n statements are true, without revealing which ones those are. Such proofs have received significant interest over the years, as they turn out to be simple enough to be constructed efficiently, while at the same time being expressive enough to be applicable to a wide variety of problems. Both the work of Cramer, Damgård and Schoenmakers and that of De Santis et al. [DDP+94] show that one can generically and *information-theoretically* transform certain separate proof systems for languages $\mathcal{L}_1, \ldots, \mathcal{L}_n$ respectively, into a single new proof system that allows for proving partial knowledge, i.e. for proving that in a given vector of statements $(x_1, \ldots, x_n)$, there is a subset of indices I of size k, such that $x_i \in \mathcal{L}_i$ for all $i \in I$. The resulting proof systems are conceptually simple and can allow for concretely efficient instantiations, but require the prover and verifier to communicate at least $\Omega(n)$ bits.

Subsequent works [GK15; AC20; ACF21; ACK21; GGHA+22] have shown how to construct proofs of partial knowledge (and more) with communication complexities that have sublinear, even logarithmic, dependencies on n, but rely on *number-theoretic hardness assumptions*, such as the discrete logarithm problem. To the best of our knowledge, despite 30 years of research, there are still no direct analogues of the transformations by Cramer, Damgård and Schoenmakers or De Santis et al., which have both an $o(n)$ communication complexity and do not require number-theoretic hardness assumptions. It seems natural to ask, whether such a transformation exists.

⋆ katharina.boudgoust@lirmm.fr
⋆⋆ mark@univariate.org

1.1 Our Contribution

In this work, we make progress towards addressing the above question. We present a new approach for transforming a large class of proof systems for language $\mathcal{L}$ into one for the language $\mathcal{L}^n_{\mathsf{OR}} := \{(x_1, \ldots, x_n) \mid \exists i \in \{1, \ldots, n\} : x_i \in \mathcal{L}\}$.[3] Our transformation produces a proof system, whose communication complexity only depends logarithmically on n and that only relies on the existence of collision-resistant hash functions. As an example, we compile the famous proof system of Goldreich, Micali, and Wigderson [GMW86] for graph 3-coloring (and thus for all of NP) into a new one for showing the validity of one out of n NP statements, while only incurring asymptotically small overheads on the communication complexity and only relying on unstructured hardness assumptions.

A key technical tool of our work, which we believe to be of independent interest, is what we call non-adaptively programmable functions (NAPs). These can be seen as pseudorandom functions, which allow for dynamically re-programming the output at an input point, which must already be fixed during key generation. Even when given the re-programmed secret key, it should not be feasible to determine where re-programming happened. We formally define this primitive and show how to construct it from one-way functions (which are implied by collision-resistant hashing).

Another technical tool of our work that may also be of independent interest, is the construction of explainable samplers for distribution that can be sampled efficiently via rejection sampling. An explainable sampler takes random coins r as input and produces a sample x from a target distribution $\mathcal{D}$. They are equipped with an explanation algorithm that, given a sample x from $\mathcal{D}$, can compute the matching random coins r. Given (x, r) it should not be possible to see, whether x was sampled and r was computed from it or vice versa.

1.2 Related Works

A plethora of existing works have studied proof systems from various perspectives. We are far from the first ones to look at minimizing the communication complexity or the required hardness assumptions. In the following, let us highlight in what ways our approach differs from prior ones.

From Number-Theoretic Assumptions. A series of works [BCC+16; BBB+18; AC20; ACK21] considers the task of designing communication-efficient proof systems from number-theoretic hardness assumptions. These approaches express the given statements as arithmetic circuit satisfiability instances and then show how any such instance can be proven with a communication complexity that is logarithmic in the circuit size. These works differ from ours in two aspects. They all rely on hardness assumptions that imply public-key cryptography, whereas our focus is on solely relying on collision-resistant hashing. Furthermore, these approaches focus on building proof systems from scratch, which can be used to generically transform an instance of $\mathcal{L}^n_{\mathsf{OR}}$ into an arithmetic circuit. While this is in principle possible, it would result in large circuits and also not make any use of a potentially given proof system for language $\mathcal{L}$. In our work, we focus on a more direct approach that transforms a given proof system for $\mathcal{L}$ into one for $\mathcal{L}^n_{\mathsf{OR}}$ efficiently.

Another line of recent works [GK15; ACF21; GGHA+22; WW22] specifically focuses on languages of the same form as $\mathcal{L}^n_{\mathsf{OR}}$. These works either aim for building proof systems for it from scratch or transforming proof systems for $\mathcal{L}$ into ones for $\mathcal{L}^n_{\mathsf{OR}}$ like we do. All of these

[3] Throughout this work, we restrict our attention to one language and showing the validity of one out of the n statements. We note, however, that this is done merely for the sake of simplicity and clarity and that our approach can easily be generalized to multiple languages and showing the validity of $k \geq 1$ statements, as we discuss in Section 4.2.

approaches crucially rely on structured hardness assumptions and it is not clear how one could modify these results to obtain something from collision-resistant hashing alone.

From One-Way or Collision-Resistant Hash Functions. Several known approaches would allow for constructing proof systems from the sole existence of one-way functions. Goldreich, Micali, and Wigderson [GMW86] show how a prover can convince a verifier that a given graph is 3-colorable. Their construction relies on the existence of commitments, which can be constructed from one-way functions [Nao90; HIL+99]. Alternatively, it is also possible to use MPC-in-the-head paradigm of Ishai, Kushilevitz, Ostrovsky, and Sahai [IKO+09], which allows for proving satisfiability of arbitrary statements, expressed as boolean or arithmetic circuits, while only relying on one-way functions. These approaches would result in communication complexities that are at least $\Omega(\sqrt{n})$, whereas we aim for a logarithmic dependency on n. Finally, using probabilistically checkable proofs [AS92; Kil92] or interactive oracle proofs [BCS16; RRR16; BBH+18], it is possible to construct proof systems from collision-resistant hashing, which have communication complexities that are poly-logarithmic in the statement size. These approaches work yet again by generically transforming an arbitrary statement into a computation expressed in their respective computational model, which depending on the type of statement can result in concretely very large communication complexities and computational overheads. Furthermore, these approaches are conceptually rather involved and complex. Contrary to them, we aim for directly transforming proof systems for $\mathcal{L}$ into ones for $\mathcal{L}^n_{\mathsf{OR}}$ via an efficient and conceptually much simpler transformation.

1.3 Technical Overview

The starting point of our work, is that of Cramer, Damgård and Schoenmakers [CDS94], which transforms proof systems known as Σ-protocols for language $\mathcal{L}$ into ones for $\mathcal{L}^n_{\mathsf{OR}}$. Before presenting our approach, let us recall what Σ-protocols are and how the aforementioned transformation works.

Languages and Relations. Throughout this work, we consider a prover that aims to convince a verifier that some statement x is in the language $\mathcal{L}$. The prover holds a witness w, which attests of x being in $\mathcal{L}$. More formally, we assume there exists an efficiently checkable relation $\mathcal{R}_{\mathcal{L}}$, such that $x \in \mathcal{L}$, if and only if there exists a witness w, such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$.

Σ-Protocols. A three-move public-coin interactive proof system that satisfies completeness, special soundness, and honest verifier zero-knowledge is known as a Σ-protocol. Here, three-move public-coin refers to the fixed communication pattern that these protocols have: the prover sends a message a to the verifier, receives a random challenge e from the verifier back, and sends a response z to the verifier, who then either accepts or rejects the proof transcript (a, e, z). Completeness dictates that an honest interaction between the prover and verifier for a statement $x \in \mathcal{L}$, results in the verifier accepting the proof. Special soundness requires that there exists an efficient extractor Ext, which can extract w with $(x, w) \in \mathcal{R}_{\mathcal{L}}$ from two accepting transcripts (a, e, z) and (a, e', z'), where $e \neq e'$. Honest verifier zero-knowledge requires that there exists a simulator Sim, which is given a uniformly random challenge e and outputs a, z, such that (a, e, z) is indistinguishable from a real interaction for $x \in \mathcal{L}$.

The Approach of Cramer, Damgård and Schoenmakers. Now assume we are given a Σ-protocol for language $\mathcal{L}$ that we would like to transform into one for language $\mathcal{L}^n_{\mathsf{OR}}$. The prover and verifier are given a vector of statements $(x_1, \ldots, x_n)$ and additionally the prover holds a witness w, such that $(x_i, w) \in \mathcal{R}_{\mathcal{L}}$ for some $i \in \{1, \ldots, n\}$. Their approach proceeds as follows: The

prover picks challenges e_j uniformly at random and uses the simulator Sim to compute the corresponding messages (a_j, z_j) for all $j \neq i$. The prover honestly compute a_i and sends the vector $(a_1, \ldots, a_n)$ to the verifier, who responds with a uniformly random challenge e. The prover computes e_i, such that $\sum_{j=1}^{n} e_j = e$, honestly computes the message z_i corresponding to the partial transcript (a_i, e_i), and sends both $(e_1, \ldots, e_n)$ and $(z_1, \ldots, z_n)$ to the verifier, who checks that all challenges sum to the right value and that all n received transcripts are accepting. This approach does not reveal which witness the prover was holding, because all challenges are uniformly random, conditioned on summing to e, and because honest verifier zero-knowledge guarantees that the transcripts produced by Sim are indistinguishable from real ones.

Our High-Level Idea. Conceptually, our work closely follows the blueprint of Cramer, Damgård and Schoenmakers, but modifies it in a way that allows us to compress all three vectors $(a_1, \ldots, a_n)$, $(e_1, \ldots, e_n)$, and $(z_1, \ldots, z_n)$, which are sent between the prover and the verifier. For this, let us make two additional assumptions. First, let us assume that the given Σ-protocol is not only honest verifier zero-knowledge, but *strongly* honest verifier zero-knowledge, which means that the simulator Sim is given a uniformly random challenge e as well as an independently chosen, uniformly random response z and is asked to compute the corresponding first message a deterministically, such that (a, e, z) is indistinguishable from a real transcript. Secondly, let us assume that we are given a privately programmable pseudorandom function $F : \mathcal{K} \times \{1, \ldots, n\} \to \{0,1\}^*$, which takes a secret key msk and point x as input and returns a random looking outputs y. We require F to be programmable in the sense that a key msk should allow for obtaining a key psk that produces the same evaluations on all inputs, apart from some chosen point $i \in \{1, \ldots, n\}$, where it returns a re-programmed value y^*. It should be privately programmable in the sense that psk should not reveal any information about where the key was re-programmed.

Equipped with these tools, our approach works as follows: The prover picks uniformly random keys msk_e and msk_z, computes $e_j := F(\mathsf{msk}_e, j)$ and $z_j := F(\mathsf{msk}_z, j)$, and uses simulator Sim to compute the corresponding messages a_j deterministically for all $j \neq i$. The prover then picks the first message a_i honestly and sends $a = H(a_1, \ldots, a_n)$ to the verifier, where H is a collision-resistant hash function. The verifier sends a challenge e to the prover, who computes e_i, such that $\sum_{j=1}^{n} e_j = e$, then computes z_i honestly. Now the prover computes keys psk_e and psk_z by re-programming keys msk_e and msk_z, such that they return e_i and z_i on input i respectively. Finally, the prover sends back psk_e and psk_z to the verifier, who expands them to the corresponding vectors $(e_1, \ldots, e_n)$ and $(z_1, \ldots, z_n)$ and then computes $(a_1, \ldots, a_n)$ using the deterministic simulator Sim. The verifier checks that all challenges sum to e, that the hash value a matches the hash of the computed vector $(a_1, \ldots, a_n)$ and that all transcripts are accepting.

Why should this be a sensible protocol on an intuitively level? The private programmability of F ensures that the verifier cannot see which location i was re-programmed. The collision-resistant hash ensures that without explicitly sending the vector $(a_1, \ldots, a_n)$, the prover is committing themselves to a fixed first message before seeing the challenge. The simulator Sim computing the first messages deterministically and the fact that simulated and real transcripts are indistinguishable, ensures that the verifier computes the correct values $a_1, \ldots, a_n$, without the prover ever having sent them explicitly.

The communication between prover and verifier is comprised of sending one hash value, one challenge, and two programmed keys. If the key sizes are logarithmic in n, then so is the communication complexity of the resulting Σ-protocol.

The Difficulty with Privately Programmable Pseudorandom Functions. To realize the above idea, we need to construct the privately programmable functions we need. Since our goal is a Σ-protocol from collision-resistant hashing, the privately programmable functions also better

be based on a similar assumption. Unfortunately, such constructions currently seem to be out of reach. The only known constructions either rely on indistinguishability obfuscation [BLW17] or lattice-based cryptography [PS18; PS20]. Luckily, we observe that we do not actually need the full power of these programmable functions. In our case, we already know during key generation at which point we would like to program the keys later on, namely at the index for which we have the witness. As it turns out, this makes constructing such functions much simpler, in fact so simple that we can construct them from just one-way functions. We call these functions non-adaptively (privately) programmable functions or NAPs and we believe that they may be of independent interest.

Constructing NAPs. To construct our new primitive, we make use of distributed point functions, originally introduced by Gilboa and Ishai [GI14]. A point function $f_{x^*} : \mathcal{X} \to \{0,1\}$ evaluates to zero on all points from domain $\mathcal{X}$, apart from a single point $x^* \in \mathcal{X}$, where it evaluates to one. A distributed point function is a pair of functions f_0, f_1 that satisfies correctness and privacy. Correctness requires that for any $x \in \mathcal{X}$, it holds that $f(x) = f_0(x) \oplus f_1(x)$. Privacy requires that f_0 and f_1 constitute an additive secret sharing of f, i.e. that neither share in isolation provides any information about which point function was secret shared. It was shown by Boyle, Gilboa, and Ishai [BGI15] that distributed point functions, where each share is of size $\mathcal{O}(\lambda \cdot \log|\mathcal{X}|)$, can be constructed from one-way functions, where λ is the computational security parameter.

Towards constructing NAPs, we make a simple, but important observation. For any $x \neq x^*$, we have that $f_0(x) \oplus f_1(x) = 0$ and thus $f_0(x) = f_1(x)$. For x^* on the other hand, we have that $f_0(x^*) \oplus f_1(x^*) = 1$ and thus $(f_0(x^*), f_1(x^*)) \in \{(0,1),(1,0)\}$. Now we can simply view f_0 as the key of our NAP (with one bit outputs) and evaluating it at point j can be done by returning $f_0(j)$. Furthermore, programming a key at x^* to value $y^* \in \{0,1\}$, can be done by returning f_b with $f_b(x^*) = y^*$ as the programmed key. Note that seeing the programmed key, is the same as seeing a single share of a 2-out-of-2 secret sharing of f and thus it does not reveal anything about the point x^*. Also, note that this approach crucially relies on the fact that the point at which programming will happen is known during key generation, as it defines the point function that will be shared. To obtain a NAP with t bits of output, we can simply append t many NAPs with single bit outputs. The resulting key size is $\mathcal{O}(t \cdot \lambda \cdot \log|\mathcal{X}|)$.

We note that the same insights we use here to construct NAPs from distributed point functions, have previously been used Hemenway et al. [HJO+16] to construct a notion they call somewhere equivocal encryption. There, the authors use their notion of an encryption scheme to allow programming the plaintext *within the security proof*. In our work on the other hand, we use NAPs and the ability to program them as part of the proof systems we construct.

Constructing NAPs for the Correct Distributions. In the discussion above, we have made an implicit assumption that is not actually valid. The NAPs we have so far constructed, return bit strings, but the second and third round messages of a given Σ-protocol may be completely different objects. As an example, in the proof system of Goldreich, Micali, and Wigderson [GMW86] for the graph isomorphism problem, the challenge e is a bit, but the prover's response z is a permutation. More generally, it may not even be the case that the response is a uniformly random element from the set of all responses.

To overcome this issue, we make use of explainable samplers, which were formally defined by Lu and Waters [LW22]. Such samplers take random coins r as input and produce outputs x that are sampled accordingly to some target distribution $\mathcal{D}$. The samplers have an associated explanation algorithm that first takes sample x from the target distribution and then finds appropriate random coins r for the sampling algorithm, i.e. coins that would produce that sample. Explainable sampler guarantee that, given (x, r), one cannot distinguish whether r was sampled and x was computed or vice versa. Given an explainable sampler for a desired target

distribution, we can combine it with our NAP that produces uniformly random looking bits, to obtain a NAP that produces pseudorandom samples from the desired target distribution.

What remains to show is that we can construct efficient explainable samplers for the distributions we care about. Lu and Waters construct explainable samplers for several classes of distribution. We extend their results and show that anything that can be sampled efficiently via a method known as rejection sampling, can also be explained. Using this result, we construct NAPs that output third round messages of well-known Σ-protocols, such as the graph 3-coloring or the graph isomorphism protocols of Goldreich, Micali, and Wigderson.

On the Strong Honest Verifier Zero-Knowledge Requirement. We note that our construction starts with a Σ-protocol that satisfies strong honest verifier zero-knowledge. Many of the existing Σ-protocols either already satisfy this property or can be made to have it via minimal changes. More generally, it was shown by Goel, Green, Hall-Andersen, and Kaptchuk [GGHA+22] that any Σ-protocol with honest verifier zero-knowledge can be transformed into one that satisfies the strong version of this property.[4] Thus our approach is in principle applicable to any Σ-protocol for which we can construct NAPs for the appropriate distributions of the second and third round messages.

Proofs vs. Arguments. Throughout the introduction and throughout the rest of this paper, we are not making an explicit distinction between proofs and arguments. Commonly, proofs provide soundness against an unbounded adversary, whereas arguments only provide soundness against a computationally bounded adversary. We note that the soundness of the Σ-protocols we construct in this work, relies on collision-resistant hash functions and thus these are arguments, not proofs. For the remainder of the paper, we use these terms interchangeably.

2 Preliminaries

Notation. We write $y \leftarrow A(x)$ to denote the output y of algorithm A, when run on input x. If A is randomized, we assume that uniformly random coins are chosen implicitly, unless stated otherwise. If we want to make random coins r explicit, we write $A(x; r)$.

For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote sampling a value from $\mathcal{D}$ and assigning the value to x. We implicitly assume that all distribution of interest in this work are efficiently samplable. We write $\mathsf{Supp}(\mathcal{D})$ to denote the set of elements that are sampled with a non-zero probability. For $a, b \in \mathbb{R}$ with $a \leq b$, we write $\mathcal{U}[a, b]$ to denote the continuous uniform distribution over the range $[a, b]$. For a set $S = \{s_1, \ldots, s_n\}$, we write $\mathcal{U}\{s_1, \ldots, s_n\}$ or $\mathcal{U}_S$ to denote the discrete uniform distribution over the set S. For a set S, we write $x \leftarrow S$ to denote sampling from the uniform distribution over S. By $(\mathcal{S}_n, \circ)$ we denote the group of permutations $\pi\colon [n] \to [n]$ with $\circ$ the composition of permutations as group operation. We use λ to denote the security parameter. For a language $\mathcal{L} \subseteq \{0,1\}^*$ we let $\mathcal{R}_\mathcal{L}$ be the corresponding relation. That is, $x \in \mathcal{L}$ if and only if there exists a witness w such that $(x, w) \in \mathcal{R}_\mathcal{L}$.

We call two distributions $\mathcal{P}$ and $\mathcal{Q}$ statistically close, if for any adversary $\mathcal{A}$, it holds that $|\Pr[1 \leftarrow \mathcal{A}(x)\colon x \leftarrow \mathcal{P}] - [1 \leftarrow \mathcal{A}(x)\colon x \leftarrow \mathcal{Q}]| \leq \mathsf{negl}(\lambda)$, and computationally close if the same holds for any PPT adversary $\mathcal{A}$.

2.1 Σ-Protocols

We recall the definition of Σ-protocols and some of their properties.

[4] In the work of Goel, Green, Hall-Andersen, and Kaptchuk [GGHA+22], the terminology *challenge-independent extended* honest verifier zero-knowledge was used, but their notion is identical to the notion of strong honest verifier zero-knowledge of Bellare and Ristov [BR08].

Definition 1 (Σ-Protocols). *A Σ-protocol is a three-move protocol with challenge space $\mathcal{E}$ and response distribution $\mathcal{Z}$ for a language $\mathcal{L}$, given by a tuple of PPT algorithms $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$, which are defined as follows:*

$(a, \mathsf{aux}) \leftarrow \mathsf{P}_1(x, w)$**:** *The commitment algorithm takes statement $x \in \mathcal{L}$ and a witness w as input and produces message a and auxiliary output aux.*

$z \leftarrow \mathsf{P}_2(\mathsf{aux}, e)$**:** *The response algorithm takes challenge $e \in \mathcal{E}$ and auxiliary input aux as input and returns response $z \in \mathsf{Supp}(\mathcal{Z})$.*

$b \leftarrow \mathsf{V}(x, a, e, z)$**:** *The verification algorithm takes statement x and transcript (a, e, z) as input and outputs bit b.*

Correctness states that during an honest execution, the verifier all but a negligible fraction of times.

Definition 2 (Correctness). *We say Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with challenge space $\mathcal{E}$ and response distribution $\mathcal{Z}$ for language $\mathcal{L}$ is* correct*, if for any $\lambda \in \mathbb{N}$, and any (x, w) with $\mathcal{R}_{\mathcal{L}}(x, w) = 1$, it holds that*

$$\Pr\left[\mathsf{V}(x, a, e, z) = 1 \colon \begin{array}{r} (a, \mathsf{aux}) \leftarrow \mathsf{P}_1(x, w) \\ e \leftarrow \mathcal{E} \\ z \leftarrow \mathsf{P}_2(\mathsf{aux}, e) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the algorithms.

Special soundness guarantees for statements outside the language, no two valid transcripts with the same first message but different challenges exist.

Definition 3 (Special Soundness). *We say Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with challenge space $\mathcal{E}$ and response distribution $\mathcal{Z}$ for language $\mathcal{L}$ is* special sound*, if there exists a PPT extractor Ext, such that for any $x \in \{0,1\}^*$, any two transcripts (a, e, z) and (a, e', z') with $\mathsf{V}(x, a, e, z) = 1$ and $\mathsf{V}(x, a, e', z') = 1$ and with $e \neq e'$, it holds that*

$$\Pr\left[\mathcal{R}_{\mathcal{L}}(x, w) = 1 \colon w \leftarrow \mathsf{Ext}(x, a, e, z, e', z')\right] = 1,$$

where the probability is taken over the random coins of the extractor.

Computational special soundness allows such pairs of transcripts to exist, but requires them to be computationally hard to find.

Definition 4 (Computational Special Soundness). *We say Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with challenge space $\mathcal{E}$ and response distribution $\mathcal{Z}$ for language $\mathcal{L}$ is* computationally special sound*, if for any $\lambda \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}}(1^\lambda) = 1] := \Pr\left[\begin{array}{c} e \neq e' \wedge x \notin \mathcal{L} \\ \mathsf{V}(x, a, e', z') = 1 \colon (x, a, e, z, e', z') \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{V}(x, a, e, z) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the experiment.

Honest verifier zero-knowledge guarantees a simulator which, on input a random challenge, can simulate valid transcripts.

Definition 5 (Honest Verifier Zero-Knowledge). *We say Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with challenge space $\mathcal{E}$ and response distribution $\mathcal{Z}$ for language $\mathcal{L}$ is* honest verifier zero-knowledge*, if there exists a PPT simulator Sim, such that for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{A}$, all $x \in \mathcal{L}$ and all witnesses w with $\mathcal{R}_{\mathcal{L}}(x, w) = 1$, it holds that*

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{Real}\Sigma}_{\mathcal{A}}(1^\lambda, x, w) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{Sim}\Sigma}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, x) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the adversary and the experiments defined in Figure 1.

Remark 1. We remark that our definition of honest verifier zero-knowledge is sometimes also called *special* honest verifier zero-knowledge.

The property below strengthens honest verifier zero-knowledge in the sense that the simulator now gets as input a random challenge and a random response and is required to be deterministic.

Definition 6 (Strong Honest Verifier Zero-Knowledge [BR08]). *We say Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with challenge space $\mathcal{E}$ and response distribution $\mathcal{Z}$ for language $\mathcal{L}$ is* strong honest verifier zero-knowledge*, if there exists a deterministic polynomial time simulator Sim, such that for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{A}$, all $x \in \mathcal{L}$ and all witnesses w with $\mathcal{R}_{\mathcal{L}}(x, w) = 1$, it holds that*

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{stReal}\Sigma}_{\mathcal{A}}(1^\lambda, x, w) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{stSim}\Sigma}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, x) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the adversary and the experiments defined in Figure 2.

$\mathsf{Expt}^{\mathsf{Real}\Sigma}_{\mathcal{A}}(1^\lambda, x, w)$

1 : $(a, \mathsf{aux}) \leftarrow \mathsf{P}_1(x, w)$
2 : $e \leftarrow \mathcal{E}$
3 : $z \leftarrow \mathsf{P}_2(\mathsf{aux}, e)$
4 : $b \leftarrow \mathcal{A}(a, e, z)$
5 : **return** b

$\mathsf{Expt}^{\mathsf{Sim}\Sigma}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, x)$

1 : $e \leftarrow \mathcal{E}$
2 : $(a, z) \leftarrow \mathsf{Sim}(e)$
3 : $b \leftarrow \mathcal{A}(a, e, z)$
4 : **return** b

Fig. 1. Honest verifier zero-knowledge.

$\mathsf{Expt}^{\mathsf{stReal}\Sigma}_{\mathcal{A}}(1^\lambda, x, w)$

1 : $(a, \mathsf{aux}) \leftarrow \mathsf{P}_1(x, w)$
2 : $e \leftarrow \mathcal{E}$
3 : $z \leftarrow \mathsf{P}_2(\mathsf{aux}, e)$
4 : $b \leftarrow \mathcal{A}(a, e, z)$
5 : **return** b

$\mathsf{Expt}^{\mathsf{stSim}\Sigma}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, x)$

1 : $e \leftarrow \mathcal{E}$
2 : $z \leftarrow \mathcal{Z}$
3 : $a \leftarrow \mathsf{Sim}(e, z)$
4 : $b \leftarrow \mathcal{A}(a, e, z)$
5 : **return** b

Fig. 2. Strong honest verifier zero-knowledge.

Remark 2. In some cases, we will assume that our Σ-protocols run in the presence of a honestly sampled common reference string. This string will, for instance, contain the description of a hash function or a commitment scheme. If a Σ-protocol requires such a string, then throughout the paper, we will assume that this string is sampled honestly at the start of any experiment and all probabilities are taken over the random coins that were used to sampled this string as well.

2.2 Distributed Point Functions [GI14]

We recall the definition of (distributed) point functions.

Definition 7 (Point Functions). *For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, a* point function $f_{x,y}$ *with domain $\mathcal{X}$ and range $\mathcal{Y}$ is defined as*

$$f_{x,y}(z) = \begin{cases} y & \text{if } z = x \\ 0 & \text{else} \end{cases}.$$

Let $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ be the set of point functions with domain $\mathcal{X}$ and range $\mathcal{Y}$.

Definition 8 (Distributed Point Functions). *A* distributed point function *for domain $\mathcal{X}$ and range $\mathcal{Y}$ is a pair of PPT algorithms $\mathsf{DPF} = (\mathsf{Share}, \mathsf{Eval})$ that are defined as follows:*

$(f_0, f_1) \leftarrow \mathsf{Share}(1^\lambda, f)$**:** *The share generation algorithm takes the security parameter λ and point function $f \in \mathcal{F}(\mathcal{X}, \mathcal{Y})$ as input and returns function shares f_0 and f_1.*

$y \leftarrow \mathsf{Eval}(f_b, x)$**:** *The evaluation algorithm takes function share f_b for $b \in \{0,1\}$ and $x \in \mathcal{X}$ as input and returns evaluation $y \in \mathcal{Y}$.*

Correctness states that combining the evaluations of both shares gives the original value.

Definition 9 (Correctness). *We say a $\mathsf{DPF} = (\mathsf{Share}, \mathsf{Eval})$ for domain $\mathcal{X}$ and range $\mathcal{Y}$, where $\mathcal{Y}$ is an abelian group with addition, is* correct*, if for any $\lambda \in \mathbb{N}$, any function $f \in \mathcal{F}(\mathcal{X}, \mathcal{Y})$ and any $z \in \mathcal{X}$, it holds that*

$$\Pr\left[\mathsf{Eval}(f_0, z) + \mathsf{Eval}(f_1, z) = f(z) \colon (f_0, f_1) \leftarrow \mathsf{Gen}(1^\lambda, f)\right] = 1.$$

Privacy guarantees that shares do not leak any information about the function they are derived from.

Definition 10 (Privacy). *We say a $\mathsf{DPF} = (\mathsf{Share}, \mathsf{Eval})$ for domain $\mathcal{X}$ and range $\mathcal{Y}$, is* private*, if for any $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$, it holds that*

$$\mathsf{Adv}^{\mathsf{priv}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Expt}^{\mathsf{priv}}_{\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Expt}^{\mathsf{priv}}_{\mathcal{A}}$ is the experiment in Fig. 3.

$\mathsf{Expt}^{\mathsf{priv}}_{\mathcal{A}}(1^\lambda)$

1 : $f^0, f^1 \leftarrow \mathcal{A}(1^\lambda)$
2 : $b, \tilde{b} \leftarrow \{0,1\}$
3 : $(f_0^b, f_1^b) \leftarrow \mathsf{Share}(1^\lambda, f^b)$
4 : $b' \leftarrow \mathcal{A}(f_{\tilde{b}}^b)$
5 : **return** $b = b'$

Fig. 3. The privacy experiment for DPFs.

$\mathsf{Expt}^{\mathsf{RealS}}_{\mathcal{A}}(1^\lambda)$

1 : $r \leftarrow \mathcal{R}$
2 : $x \leftarrow \mathsf{Sample}(1^\lambda; r)$
3 : $b \leftarrow \mathcal{A}(r, x)$
4 : **return** b

$\mathsf{Expt}^{\mathsf{ExplainS}}_{\mathcal{A}}(1^\lambda)$

1 : $r' \leftarrow \mathcal{R}$
2 : $x \leftarrow \mathsf{Sample}(1^\lambda; r')$
3 : $r \leftarrow \mathsf{Explain}(1^\lambda, x)$
4 : $b \leftarrow \mathcal{A}(r, x)$
5 : **return** b

Fig. 4. The explainability experiments for samplers.

We use the following result which shows that one can obtain DPFs from pseudorandom generators, which can themselves be obtained from one-way functions [HIL+99].

Theorem 1 ([BGI15]). *Let $\lambda, \ell \in \mathbb{N}$. Assuming the existence of a pseudorandom generators, there exists a correct and private DPF for domain $\mathcal{X} = \{0,1\}^\ell$ and range $\mathcal{Y} = \{0,1\}$ with shares of bit size $\mathcal{O}(\lambda \log|\mathcal{X}|)$.*

2.3 Explainable Samplers [LW22]

Our definition of explainable samplers slightly differs from that of Lu and Waters [LW22]. Their definition assumes that the random coins provided to the sampling algorithm come from a uniformly random distribution. In our definition, we allow the coins to come from other distributions. In their definition, a separate precision parameter specifies how "well" the explain algorithm works. In our definition, we will not have a separate precision parameter, but instead assume that the advantage of the adversary in each security experiment will be negligible in the same security parameter λ.

Definition 11 (Samplers). *Let $\lambda \in \mathbb{N}$. A sampler for distribution $\mathcal{D} = \mathcal{D}(\lambda)$ with randomness distribution $\mathcal{R} = \mathcal{R}(\lambda)$ is a pair of PPT algorithms $\mathsf{ES} = (\mathsf{Sample}, \mathsf{Explain})$ that are defined as follows:*

$x \leftarrow \mathsf{Sample}(1^\lambda)$**:** *The sampling algorithm takes the security parameter λ as input and returns a sample x.*

$r \leftarrow \mathsf{Explain}(1^\lambda, x)$**:** *The explaining algorithm takes security parameter λ and sample $x \in \mathsf{Supp}(\mathcal{D})$ as input and returns random coins r.*

An explainable sampler should be correct in the sense that the samples returned by Sample should be statistically close to samples from the real distribution.

Definition 12 (Correctness). *Let $\lambda \in \mathbb{N}$. A sampler for distribution $\mathcal{D} = \mathcal{D}(\lambda)$ with randomness distribution $\mathcal{R} = \mathcal{R}(\lambda)$ is* correct*, if for any λ and any adversary $\mathcal{A}$, it holds that*

$$\left|\Pr\left[\mathcal{A}(x) = 1 : x \leftarrow \mathsf{Sample}(1^\lambda; r);\ r \leftarrow \mathcal{R}\right] - \Pr\left[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{D}\right]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of all algorithms.

An explainable sampler should be explainable in the sense that first sampling an element $x \in \mathsf{Supp}(\mathcal{D})$ and then finding random coins $r \in \mathsf{Supp}(\mathcal{R})$, such that $\mathsf{Sample}(1^\lambda; r) = x$, should be statistically indistinguishable from first sampling $r \leftarrow \mathcal{R}$ and then computing $x \leftarrow \mathsf{Sample}(1^\lambda; r)$.

Definition 13 (Explainability). *Let $\lambda \in \mathbb{N}$. A sampler for distribution $\mathcal{D} = \mathcal{D}(\lambda)$ with randomness distribution $\mathcal{R} = \mathcal{R}(\lambda)$ is* explainable*, if for any λ and adversary $\mathcal{A}$, it holds that*

$$\left|\Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealS}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ExplainS}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealS}}$ and $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ExplainS}}$ are the experiments defined in Figure 4 and the probability is taken over the random coins of all algorithms.

3 Non-Adaptively Privately Programmable Functions (NAPs)

This section introduces our new cryptographic primitive, which we call non-adaptive programmable functions (NAPs). These NAPs will be the main tool that allows us to construct our compressed OR-proofs in the next section.

3.1 Definitions

We start by introducing the syntax of NAPs. On a high level, they can be seen as keyed pseudorandom functions, which allow for programming the secret key.

Definition 14 (Non-Adaptively Programmable Functions). *A* non-adaptively programmable function *with domain $\mathcal{X}$ and range $\mathcal{Y}$ with output distribution $\mathcal{D}_{\mathcal{Y}}$ is a tuple of PPT algorithms* $\mathsf{NAP} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Prog}, \mathsf{PEval})$ *that are defined as follows:*

$\mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*)$**:** *They key generation algorithm takes the security parameter 1^λ and evaluation point $x^* \in \mathcal{X}$ as input and returns a master secret key* msk.

$y \leftarrow \mathsf{Eval}(\mathsf{msk}, x)$**:** *The evaluation algorithm takes the master secret key* msk *and an evaluation point $x \in \mathcal{X}$ and returns an evaluation $y \in \mathcal{Y}$.*

$\mathsf{psk} \leftarrow \mathsf{Prog}(\mathsf{msk}, y^*)$**:** *The programming algorithm takes as input the master secret key* msk *and an evaluation $y^* \in \mathcal{Y}$, and returns a programmed secret key* psk.

$y \leftarrow \mathsf{PEval}(\mathsf{psk}, x)$**:** *The evaluation algorithm for programmed keys takes the programmed secret key* psk *and an evaluation point $x \in \mathcal{X}$ as input and returns the evaluation $y \in \mathcal{Y}$.*

We now define two properties that we would like our NAPs to satisfy. The first one is correctness, which requires that programming works as one would expect. At the programmed location, the programmed key should return the programmed output value and on all other inputs, it should return the same values as the original master secret key.

Definition 15 (Correctness). *We say* NAP *for domain $\mathcal{X}$ and output distribution $\mathcal{D}_{\mathcal{Y}}$ is* correct*, if for all $\lambda \in \mathbb{N}$, all $x^* \in \mathcal{X}$ we have*

$$\Pr\left[\begin{array}{ll} \mathsf{PEval}(\mathsf{psk}, x^*) = y^* & \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ \wedge\, \forall x \in \mathcal{X} \setminus \{x^*\} & : \quad y^* \leftarrow \mathcal{D}_{\mathcal{Y}} \\ \mathsf{PEval}(\mathsf{psk}, x) = \mathsf{Eval}(\mathsf{msk}, x) & \mathsf{psk} \leftarrow \mathsf{Prog}(\mathsf{msk}, y^*) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of all algorithms and the choice of y^*.

The second property is private programmability, which requires the programmed key to hide the location at which it was programmed. This should hold in a strong sense, where the adversary is allowed to choose location x^*, is then either given a key programmed at that location or a simulated key, and should not be able to tell in which of those two cases they are.

Definition 16 (Private Programmability). *We say* NAP *for domain $\mathcal{X}$ and output distribution $\mathcal{D}_{\mathcal{Y}}$ is* privately programmable*, if there is a PPT simulator* Sim *such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$, it holds that*

$$\left|\Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealPP}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IdealPP}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where RealPP *and* $\mathsf{IdealPP}$ *are the experiments defined in Figure 5.*

3.2 Constructions

We start with building our first NAP whose output are uniformly random bits, from distributed point functions.

Theorem 2. *Let* $\mathsf{DPF} = (\mathsf{Share}, \mathsf{Eval})$ *be a correct and private distributed point function for domain $\mathcal{X}$ and range $\mathcal{Y} := \{0,1\}$. Then the construction* $\mathsf{NAP} = (\mathsf{Gen}, \mathsf{Prog}, \mathsf{Eval}, \mathsf{PEval})$ *in Figure 6 is a correct and privately programmable NAP for domain $\mathcal{X}$ and output distribution $\mathcal{U}_{\mathcal{Y}}$.*

Proof. To prove the theorem statement, let us consider each property of a NAP separately:

$\mathsf{Expt}^{\mathsf{RealPP}}_{\mathcal{A}}(1^\lambda)$	$\mathsf{Expt}^{\mathsf{IdealPP}}_{\mathcal{A}}(1^\lambda)$
$x^* \leftarrow \mathcal{A}(1^\lambda)$	$x^* \leftarrow \mathcal{A}(1^\lambda)$
$y^* \leftarrow \mathcal{D}_{\mathcal{Y}}$	$\mathsf{psk} \leftarrow \mathsf{Sim}(1^\lambda)$
$\mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*)$	$b \leftarrow \mathcal{A}(\mathsf{psk})$
$\mathsf{psk} \leftarrow \mathsf{Prog}(\mathsf{msk}, y^*)$	**return** b
$b \leftarrow \mathcal{A}(\mathsf{psk})$	
return b	

Fig. 5. The private programmability experiment for NAPs.

Correctness. Note that $\mathcal{Y}$ with the xor-operation $\oplus$ defines an abelian group. We observe that for any $u, v \in \{0,1\}$ the two simple implications hold. If $u \oplus v = 0$, then $u = v$ and if $u \oplus v = 1$, then $u \neq v$ and thus in this case $(u, v) \in \{(1,0),(0,1)\}$. Now for any $x^* \in \mathcal{X}$, let

$$f(x) := \begin{cases} 1 \text{ if } x = x^* \\ 0 \text{ else} \end{cases}$$

and let $(f_0, f_1) \leftarrow \mathsf{DPF.Share}(1^\lambda, f)$. Correctness of the distributed point function tells us that for any $x \in \mathcal{X}$ with $x \neq x^*$, it holds that $\mathsf{DPF.Eval}(f_0, x) \oplus \mathsf{DPF.Eval}(f_1, x) = 0$ and thus $\mathsf{DPF.Eval}(f_0, x) = \mathsf{DPF.Eval}(f_1, x)$, whereas for x^* we have that $\mathsf{DPF.Eval}(f_0, x^*) \oplus \mathsf{DPF.Eval}(f_1, x^*) = 1$ and thus $(\mathsf{DPF.Eval}(f_0, x^*), \mathsf{DPF.Eval}(f_1, x^*)) \in \{(0,1),(1,0)\}$. Since for any $y^* \in \{0,1\}$, there exists a $b \in \{0,1\}$ with $\mathsf{DPF.Eval}(f_b, x^*) = y^*$, the correctness of our NAP construction follows.

Private Programmability. Fix an arbitrary $\tilde{x} \in \mathcal{X}$ and let f be the point function that evaluates to one at $\tilde{x}$. We define $\mathsf{Sim}(1^\lambda)$ to be the algorithm that generates $(f_0, f_1) \leftarrow \mathsf{Share}(1^\lambda, f)$ and then returns $f_{b'}$ for a uniformly random b'. Note that returning a uniformly random share is the same as programming the output at $\tilde{x}$ to a uniformly random value. Let $\mathcal{A}$ be an arbitrary PPT adversary with

$$\epsilon := \left| \Pr\left[\mathsf{Expt}^{\mathsf{IdealPP}}_{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{RealPP}}_{\mathcal{A}}(1^\lambda) = 1\right] \right|.$$

We construct a PPT adversary $\mathcal{B}$ against the privacy property of the distributed point function, i.e., an adversary that has advantage $\epsilon/2$ in the experiment $\mathsf{Expt}^{\mathsf{priv}}_{\mathcal{B}}(1^\lambda)$. The reduction $\mathcal{B}$ works as follows: They initialize $\mathcal{A}$ with fresh random coins, provide them with the security parameter and obtain x^*. They then define

$$f^0(x) := \begin{cases} 1 \text{ if } x = x^* \\ 0 \text{ else} \end{cases} \quad \text{and} \quad f^1(x) := \begin{cases} 1 \text{ if } x = \tilde{x} \\ 0 \text{ else} \end{cases}$$

and provide f^0 and f^1 to their challenger, who returns f^b_0. Adversary $\mathcal{B}$ forwards f^b_0 to $\mathcal{A}$ and then returns whatever bit $\mathcal{A}$ returns.

We now observe that when $b = 0$ in the privacy experiment of the distributed point function, then $\mathcal{B}$ perfectly simulates $\mathsf{Expt}^{\mathsf{RealPP}}_{\mathcal{A}}(1^\lambda)$ towards $\mathcal{A}$, as they are receiving $f^0_{\tilde{b}}$ for a uniformly random $\tilde{b} \in \{0,1\}$ as expected. When $b = 1$, then $\mathcal{B}$ perfectly simulates $\mathsf{Expt}^{\mathsf{IdealPP}}_{\mathcal{A}}(1^\lambda)$ towards $\mathcal{A}$, they obtain the expected $f^1_{\tilde{b}}$ for uniformly random $\tilde{b}$. Let $\mathsf{Expt}^{\mathsf{priv},b}_{\mathcal{A}}(1^\lambda)$ be the privacy experiment for the distributed point function, where the challenger chooses bit b. From the above observations, we can conclude that

$$2 \cdot \mathsf{Adv}^{\mathsf{priv}}(\mathcal{B}) = \left| \Pr\left[\mathsf{Expt}^{\mathsf{priv}}_{\mathcal{B}}(1^\lambda) = 1\right] - \frac{1}{2} \right| \cdot 2$$

$$
\begin{aligned}
&= \left| \frac{1}{2} \cdot \Pr\left[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{priv},0}(1^\lambda) = 1\right] + \frac{1}{2} \cdot \Pr\left[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{priv},1}(1^\lambda) = 1\right] - \frac{1}{2} \right| \cdot 2 \\
&= \left| \Pr\left[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{priv},1}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{priv},0}(1^\lambda) = 0\right] \right| \\
&= \left| \Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IdealPP}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealPP}}(1^\lambda) = 1\right] \right| = \epsilon.
\end{aligned}
$$

Since the distributed point function is private, it means that $2 \cdot \mathsf{Adv}^{\mathsf{priv}}(\mathcal{B})$ is negligible in λ and thus so is ϵ. □

Concatenating the outputs of several NAPs with one bit outputs, we can obtain a NAP with multiple output bits. This is captured formally in the next theorem statement, whose proof uses a standard hybrid argument and we thus defer to Section A.2.

Theorem 3. *Let $\lambda, t \in \mathbb{N}$ with $t \in \mathsf{poly}(\lambda)$. Let NAP' be a correct and privately programmable NAP for domain $\mathcal{X}$ and output distribution $\mathcal{U}\{0,1\}$. Then there exists a correct and privately programmable NAP for domain $\mathcal{X}$ and output distribution $\mathcal{U}\{0,1\}^t$.*

$\mathsf{Gen}(1^\lambda, x^*)$

1 : $f(x) := \begin{cases} 1 \text{ if } x = x^* \\ 0 \text{ else} \end{cases}$
2 : $(f_0, f_1) \leftarrow \mathsf{DPF.Share}(1^\lambda, f)$
3 : $\mathsf{msk} := (f_0, f_1, x^*)$
4 : **return** msk

$\mathsf{Eval}(\mathsf{msk}, x)$

1 : parse msk as (f_0, f_1, x^*)
2 : **return** $\mathsf{DPF.Eval}(f_0, x)$

$\mathsf{Prog}(\mathsf{msk}, y^*)$

1 : parse msk as (f_0, f_1, x^*)
2 : **if** $\mathsf{DPF.Eval}(f_0, x^*) = y^*$:
3 : $\quad \mathsf{psk} := f_0$
4 : **else**
5 : $\quad \mathsf{psk} := f_1$
6 : **return** psk

$\mathsf{PEval}(\mathsf{psk}, x)$

1 : parse psk as f_b
2 : **return** $\mathsf{DPF.Eval}(f_b, x)$

Fig. 6. NAP for uniform single bit outputs.

$\mathsf{Gen}(1^\lambda, x^*)$

1 : $\mathsf{msk} \leftarrow \mathsf{NAP}'.\mathsf{Gen}(1^\lambda, x^*)$
2 : **return** msk

$\mathsf{Eval}(\mathsf{msk}, x)$

1 : $r \leftarrow \mathsf{NAP}'.\mathsf{Eval}(\mathsf{msk}, x)$
2 : **return** $\mathsf{ES.Sample}(1^\lambda; r)$

$\mathsf{Prog}(\mathsf{msk}, y^*)$

1 : $r^* \leftarrow \mathsf{ES.Explain}(1^\lambda, y^*)$
2 : $\mathsf{psk} \leftarrow \mathsf{NAP}'.\mathsf{Prog}(\mathsf{msk}, r^*)$
3 : **return** psk

$\mathsf{PEval}(\mathsf{psk}, x)$

1 : $r \leftarrow \mathsf{NAP}'.\mathsf{PEval}(\mathsf{psk}, x)$
2 : **return** $\mathsf{ES.Sample}(1^\lambda; r)$

Fig. 7. NAP for explainable output distributions.

Next, we show that a NAP for one output distribution can be transformed into a NAP for a different distribution by using an appropriate explainable sampler. A bit more concretely, given a NAP with output distribution $\mathcal{R}$ and an explainable sampler with randomness distribution $\mathcal{R}$ and output distribution $\mathcal{X}$, we obtain a NAP with output distribution $\mathcal{X}$. Towards this goal, we use the NAP's output be the input random coins of the sampling algorithm and the output of the sampling algorithm is then defined to be the output of our newly constructed NAP. Programmability of the new NAP will follow from the programmability of the underlying NAP

and the explainability of the used sampler. We defer the proof of the following theorem to Section A.3.

Theorem 4. *Let $\lambda \in \mathbb{N}$. Let NAP' be a correct and privately programmable NAP for domain $\mathcal{X}$ and output distribution $\mathcal{R}$. Let $\mathsf{ES} = (\mathsf{Sample}, \mathsf{Explain})$ be a correct and explainable sampler with randomness distribution $\mathcal{R}$ and output distribution $\mathcal{D}$. Then the construction $\mathsf{NAP} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Prog}, \mathsf{PEval})$ in Figure 7 is a correct and privately programmable NAP for domain $\mathcal{X}$ and output distribution $\mathcal{D}$.*

3.3 The Sizes of NAP Keys

At this point, let us take a moment to reflect on the sizes of our NAP keys. In the construction from Theorem 2, both master secret and programmed keys are composed of distributed point function shares. Using the construction from Theorem 1, this would result in a NAP key size of $\mathcal{O}(\lambda \log|\mathcal{X}|)$. Plugging this construction into the multiple bit construction from Theorem 3, we get a NAP for uniformly random t-bit outputs and a key size of $\mathcal{O}(t\lambda \log|\mathcal{X}|)$. Alternatively, using this construction in combination with Theorem 4 and an explainable sampler with a randomness distribution that is the uniform distribution over t-bit strings and output distribution $\mathcal{D}$, we get a NAP with key size $\mathcal{O}(t\lambda \log|\mathcal{X}|)$ and output distribution $\mathcal{D}$.

4 Compressing OR-Proofs from NAPs

We now present our main contribution: compressed OR-proofs for Σ-protocols using NAPs. We start by recalling the definition of OR languages, then present our construction with security proofs in Section 4.1, sketch how to extend the OR-proof to the more general k-out-of-n setting in Section 4.2 and conclude with providing two concrete instantiations in Section 4.3.

Definition 17 (OR Languages $\mathcal{L}_{\mathsf{OR}}$). *Let $n \in \mathbb{N}$ and let $\mathcal{L}$ be a language. We define*

$$\mathcal{L}^n_{\mathsf{OR}} := \{(x_1, \ldots, x_n) \subset \{0,1\}^* \mid \exists i \in [n] : x_i \in \mathcal{L}\}$$

to be the vector of statements of length n, where at least one entry is in the language $\mathcal{L}$.

4.1 Construction

Theorem 5. *Let $\lambda, n \in \mathbb{N}$. Let $\mathcal{H} = \mathcal{H}(\lambda)$ be a family of collision-resistant hash functions, mapping from $\{0,1\}^*$ to $\{0,1\}^{\Theta(\lambda)}$. Let Σ-protocol $(\mathsf{P}'_1, \mathsf{P}'_2, \mathsf{V}')$ with challenge space $\mathcal{E}$, which is an abelian group equipped with addition, and response distribution $\mathcal{Z}$ for language $\mathcal{L}$ be correct, special sound, and strong honest verifier zero-knowledge. Let Sim' be the corresponding strong honest verifier zero-knowledge simulator. Let $\mathsf{NAP}_{\mathcal{E}}$ be a correct and privately programmable NAP for domain $[n]$ and output distribution $\mathcal{U}_{\mathcal{E}}$. Let $\mathsf{NAP}_{\mathcal{Z}}$ be a correct and privately programmable NAP for domain $[n]$ and output distribution $\mathcal{Z}$. Then the construction from Figure 8 is a correct, computationally special sound, honest verifier zero-knowledge sigma protocol for the language $\mathcal{L}^n_{\mathsf{OR}}$ in the common reference string model.*

Proof. To prove the theorem statement, we need to show correctness, computational special-soundness, and honest verifier zero-knowledge. In the following, let us look at each of these separately.

Correctness. The fact that our protocol is correct (with overwhelming probability), follows by inspection.

$\mathsf{P}_1\left((x_1,\ldots,x_n),(i,w)\right)$

$\mathsf{msk}_e \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Gen}(1^\lambda, i)$
$\mathsf{msk}_z \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Gen}(1^\lambda, i)$
for $j \in [n] \setminus \{i\}$:
  $e_j \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Eval}(\mathsf{msk}_e, j)$
  $z_j \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Eval}(\mathsf{msk}_z, j)$
  $a_j \leftarrow \mathsf{Sim}'(e_j, z_j)$
$(a_i, \mathsf{aux}_i) \leftarrow \mathsf{P}_1'(x_i, w)$
$a \leftarrow H(a_1,\ldots,a_n)$
$\mathsf{aux} = (\mathsf{msk}_e, \mathsf{msk}_z, \mathsf{aux}_i)$
return (a, aux)

$\mathsf{P}_2(\mathsf{aux}, e)$

parse aux as $(\mathsf{msk}_e, \mathsf{msk}_z, \mathsf{aux}_i)$
for $j \in [n] \setminus \{i\}$:
  $e_j \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Eval}(\mathsf{msk}_e, j)$
$e_i := e - \sum_{j \neq i} e_j$
$z_i \leftarrow \mathsf{P}_2'(\mathsf{aux}_i, e_i)$
$\mathsf{psk}_e \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Prog}(\mathsf{msk}_e, e_i)$
$\mathsf{psk}_z \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Prog}(\mathsf{msk}_z, z_i)$
return $(\mathsf{psk}_e, \mathsf{psk}_z)$

$\mathsf{V}(a, e, z)$

parse z as $(\mathsf{psk}_e, \mathsf{psk}_z)$
for $j \in [n]$:
  $e_j \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{PEval}(\mathsf{psk}_e, j)$
  $z_j \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{PEval}(\mathsf{psk}_z, j)$
  $a_j \leftarrow \mathsf{Sim}'(e_j, z_j)$
if $\left(\sum_{j=1}^{n} e_j \neq e\right) \vee (H(a_1,\ldots,a_n) \neq a)$:
  return 0
if $\exists j \in [n]$ s.t. $\mathsf{V}'(x_j, a_j, e_j, z_j) = 0$:
  return 0
return 1

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

$H \leftarrow \mathcal{H}$
return $\mathsf{crs} := H$

Fig. 8. OR-proof from NAPs.

Computational Special Soundness. Let $\mathcal{A}$ be a PPT adversary, such that

$$\Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}^n_{\mathsf{OR}}}(1^\lambda) = 1] = \epsilon.$$

Recall that the winning condition of the experiment requires $\mathcal{A}$ to return a vector (x, a, e, z, e', z'), such that both $\mathsf{V}(x, a, e, z) = 1$ and $\mathsf{V}(x, a, e', z') = 1$, while at the same time $e \neq e'$ and $x \notin \mathcal{L}^n_{\mathsf{OR}}$.

During the verification of (x, a, e, z) and (x, a, e', z'), at step 2 of V in Figure 8, for each $j \in [n]$, the verifier computes transcripts (a_j, e_j, z_j) and (a'_j, e'_j, z'_j) respectively. Let COLL be the event that $H(a_1, \ldots, a_n) = H(a'_1, \ldots, a'_n)$, but $(a_1, \ldots, a_n) \neq (a'_1, \ldots, a'_n)$. We observe that

$$\begin{aligned}
&\Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}^n_{\mathsf{OR}}}(1^\lambda) = 1] \\
&= \Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}^n_{\mathsf{OR}}}(1^\lambda) = 1 \mid \mathsf{COLL}] \cdot \Pr[\mathsf{COLL}] + \Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}^n_{\mathsf{OR}}}(1^\lambda) = 1 \mid \neg\mathsf{COLL}] \cdot \Pr[\neg\mathsf{COLL}] \\
&\leq \Pr[\mathsf{COLL}] + \Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}^n_{\mathsf{OR}}}(1^\lambda) = 1 \mid \neg\mathsf{COLL}].
\end{aligned}$$

From the collision-resistance of $\mathcal{H}$, it follows that $\Pr[\mathsf{COLL}] \leq \mathsf{negl}(\lambda)$. More precisely, let $\mathcal{B}$ be an adversary against the collision-resistance of $\mathcal{H}$. Given H, the adversary $\mathcal{B}$ sets $\mathsf{crs} := H$ and honestly simulates the computational special soundness experiment towards $\mathcal{A}$. When $\mathcal{A}$ outputs (x, a, e, z, e', z'), we let $\mathcal{B}$ compute the corresponding vectors $(a_1, \ldots, a_n)$ and $(a'_1, \ldots, a'_n)$ and return them in their experiment. It is easy to see that $\mathcal{B}$ wins, whenever COLL happens and thus $\Pr[\mathsf{COLL}] \leq \mathsf{negl}(\lambda)$.

Now let us assume that COLL did not happen. Then, since the verifier checks $a = H(a_1, \ldots, a_n)$ and $a = H(a'_1, \ldots, a'_n)$ respectively in step 6 of V, and since both verifications are successful, it follows that $(a_1, \ldots, a_n) = (a'_1, \ldots, a'_n)$. However, note that $e \neq e'$ and that the verifier also checks that $\sum_{j=1}^n e_j = e$ and $\sum_{j=1}^n e'_j = e'$ respectively. It must therefore exist an index $j^* \in [n]$, such that $e_{j^*} \neq e'_{j^*}$. Since all transcripts are accepting in step 8 of V, it follows that there exists an x_{j^*} and two accepting transcripts $(a_{j^*}, e_{j^*}, z_{j^*})$ and $(a_{j^*}, e'_{j^*}, z'_{j^*})$, which agree in their first messages, but have different second messages. By assumption, the Σ-protocol $(\mathsf{P}'_1, \mathsf{P}'_2, \mathsf{V}')$ is special sound and thus it must either hold that $x_{j^*} \in \mathcal{L}$ (as the extractor from special soundness successfully extracts a witness for x_{j^*}) and thus $x \in \mathcal{L}^n_{\mathsf{OR}}$ or

$$\Pr[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\mathcal{L}^n_{\mathsf{OR}}}(1^\lambda) = 1 \mid \neg\mathsf{COLL}] = 0.$$

We can conclude that the adversary's success probability ϵ is negligible in λ.

Honest Verifier Zero-Knowledge. To show honest verifier zero-knowledge, we need to provide a simulator Sim, such that for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{A}$, all $x \in \mathcal{L}^n_{\mathsf{OR}}$ and its witnesses w, it holds that

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{Real}\Sigma}_{\mathcal{A}}(1^\lambda, x, w) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{Sim}\Sigma}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, x) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Our simulator is depicted in Figure 9.

What remains to do is to argue that our constructed simulator produces transcripts that are indistinguishable from real ones. Since both $\mathsf{NAP}_{\mathcal{E}}$ and $\mathsf{NAP}_{\mathcal{Z}}$ are privately programmable, there exist simulators $\mathsf{Sim}_{\mathcal{E}}$ and $\mathsf{Sim}_{\mathcal{Z}}$ respectively. Since honest verifier zero-knowledge only has to hold for $x \in \mathcal{L}^n_{\mathsf{OR}}$, there exists an index $i \in [n]$ with $x_i \in \mathcal{L}$. We consider the following sequence of hybrids.

Let Hybrid_0 be the experiment $\mathsf{Expt}^{\mathsf{Sim}\Sigma}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, x)$. Let Hybrid_1 be identical to Hybrid_0, apart from how the challenge e is chosen. Instead of letting the challenger pick e and provide it to the simulator, we now let the simulator pick e_1 uniformly at random and define $e = \sum_{j=1}^n e_j$. Since the challenge space $\mathcal{E}$ is an abelian group, it follows that it makes no difference, whether we

$\mathsf{Sim}(e)$

$\mathsf{msk}_e \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Gen}(1^\lambda, 1)$

$\mathsf{msk}_z \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Gen}(1^\lambda, 1)$

for $j \in [n] \setminus \{1\}$:

  $e_j \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Eval}(\mathsf{msk}_e, j)$

  $z_j \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Eval}(\mathsf{msk}_z, j)$

$e_1 := e - \sum_{j=2}^{n} e_j$

$z_1 \leftarrow \mathcal{Z}$

$\mathsf{psk}_e \leftarrow \mathsf{NAP}.\mathsf{Prog}(\mathsf{msk}_e, e_1)$

$\mathsf{psk}_z \leftarrow \mathsf{NAP}.\mathsf{Prog}(\mathsf{msk}_z, z_1)$

for $j \in [n]$:

  $a_j \leftarrow \mathsf{Sim}'(e_j, z_j)$

$a := H(a_1, \ldots, a_n)$

$z := (\mathsf{psk}_e, \mathsf{psk}_z)$

return (a, z)

Fig. 9. Honest verifier zero-knowledge simulator for the OR-proof.

first pick e_1 or e and thus the two hybrids are perfectly indistinguishable from the adversary's perspective.

Let Hybrid_2 be identical to Hybrid_1, apart from how msk_e is chosen. Instead of computing it honestly, we directly compute $\mathsf{psk}_e \leftarrow \mathsf{Sim}_{\mathcal{E}}(1^\lambda)$ and define $e_1 := \mathsf{NAP}_{\mathcal{E}}.\mathsf{PEval}(\mathsf{psk}_e, 1)$. Indistinguishability of the two hybrids follows from the private programmability of $\mathsf{NAP}_{\mathcal{E}}$. More concretely, we use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the private programmability of $\mathsf{NAP}_{\mathcal{E}}$ as follows: Initially, $\mathcal{B}$ outputs $x^* = 1$ and receives psk from the challenger. For $j \in [n]$, adversary $\mathcal{B}$ defines $e_j \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{PEval}(\mathsf{psk}, j)$. Next, $\mathcal{B}$ computes the values $\mathsf{psk}_z, z_1, e, a$ the same way as Hybrid_1 would and defines $z := (\mathsf{psk}, \mathsf{psk}_z)$. Finally, $\mathcal{B}$ calls $\mathcal{A}$ on input (a, e, z). When $\mathcal{A}$ returns bit b, we let $\mathcal{B}$ output the same bit. If $\mathcal{B}$ was in $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IdealPP}}(1^\lambda)$, then the view of $\mathcal{A}$ is identical to Hybrid_2. If $\mathcal{B}$ was in $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealPP}}(1^\lambda)$, then the view of $\mathcal{A}$ is identical to Hybrid_1. Thus by private programmability, indistinguishability of the two hybrids follows.

Let Hybrid_3 be identical to Hybrid_2, apart from how msk_z is chosen. Instead of computing it honestly, we directly compute $\mathsf{psk}_z \leftarrow \mathsf{Sim}_{\mathcal{Z}}(1^\lambda)$ and define $z_1 := \mathsf{NAP}_{\mathcal{Z}}.\mathsf{PEval}(\mathsf{psk}_z, 1)$. Indistinguishability of Hybrid_2 and Hybrid_3 follows from the private programmability of $\mathsf{NAP}_{\mathcal{Z}}$, similarly to the argument made for the previous pair of hybrids.

Let Hybrid_4 be identical to Hybrid_3, apart from now choosing $\mathsf{msk}_z \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Gen}(1^\lambda, i)$, sampling $z_i \leftarrow \mathcal{Z}$, and computing $\mathsf{psk}_z \leftarrow \mathsf{NAP}_{\mathcal{Z}}.\mathsf{Prog}(\mathsf{msk}_z, z_i)$. Here, $i \in [n]$ is the index such that $x_i \in \mathcal{L}$. Indistinguishability of Hybrid_3 and Hybrid_4 follows from the private programmability of $\mathsf{NAP}_{\mathcal{Z}}$.

Let Hybrid_5 be identical to Hybrid_4, apart from now choosing $\mathsf{msk}_e \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Gen}(1^\lambda, i)$, sampling $e_i \leftarrow \mathcal{E}$, and computing $\mathsf{psk}_e \leftarrow \mathsf{NAP}_{\mathcal{E}}.\mathsf{Prog}(\mathsf{msk}_e, e_i)$. Indistinguishability of Hybrid_4 and Hybrid_5 follows from the private programmability of $\mathsf{NAP}_{\mathcal{E}}$.

Let Hybrid_6 be identical to Hybrid_5, apart from now choosing $e \leftarrow \mathcal{E}$ and defining $e_i \leftarrow \sum_{j \neq i} e_j$. Indistinguishability of Hybrid_5 and Hybrid_6 follows from the fact that $\mathcal{E}$ is an abelian group. At this point, we arrived at a hybrid that is identical to the original simulator, but using index i, instead of index 1.

Let Hybrid_7 now be a real execution of the prover using witness (i, w), where $(x_i, w) \in \mathcal{R}_{\mathcal{L}}$. Indistinguishability of Hybrid_6 and Hybrid_7 follows from the strong honest verifier zero-knowledge property of Σ. More concretely, we use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the strong honest verifier zero-knowledge property of Σ as follows: Initially, $\mathcal{B}$ receives (a', e', z') from the challenger. They set $a_i = a'$, $e_i = e'$ and $z_i = z'$. Then they compute msk_e, msk_z as well as (a_j, e_j, z_j) for $j \neq i$ as specified by Hybrid_6 (which is the same in Hybrid_7). They set $a = H(a_1, \ldots, a_n)$, program both NAPs at the corresponding entries of e_i and z_i to derive programmed keys psk_e and psk_z, defining $z = (\mathsf{psk}_e, \mathsf{psk}_z)$. Again, this process is the same in both hybrids. Finally, $\mathcal{B}$ sets $e = \sum_{j=1}^{n} e_j$ and forwards (a, e, z) to $\mathcal{A}$. On output bit b by $\mathcal{A}$, we let $\mathcal{B}$ forward the bit as their output. Note that $\mathcal{E}$ is an abelian group, so e again has the correct distribution. If $\mathcal{B}$ was in $\mathsf{Expt}^{\mathsf{stSim}\Sigma}_{\mathcal{B},\mathsf{Sim}'}(1^\lambda, x_i)$, then the view of $\mathcal{A}$ is identical to Hybrid_6. If $\mathcal{B}$ was in $\mathsf{Expt}^{\mathsf{stReal}\Sigma}_{\mathcal{B},\mathsf{Sim}}(1^\lambda, x_i, w)$, then the view of $\mathcal{A}$ is identical to Hybrid_7 (which is identical to the real experiment of strong honest verifier zero-knowledge). Thus by strong honest verifier zero-knowledge, the two hybrids are indistinguishable.

Having shown correctness, computational special soundness, and honest verifier zero-knowledge, concludes the proof. □

4.2 Extensions

Theorem 5 allows for showing that one out of n statements is in the language $\mathcal{L}$. More generally, it may be desirable to show that there exists a subset I of size k, such that $x_i \in \mathcal{L}_i$ for all $i \in I$. That is, we would like to show that multiple statements are valid in the a setting where we deal with multiple languages. Let us shortly outline, how our approach can easily be extended to this setting.

Showing the validity of k out of n statements for one language $\mathcal{L}$. To construct such proofs of partial knowledge, we can again directly follow the blueprint of Cramer, Damgård and Schoenmakers [CDS94]. If the prover has witnesses for k statements, then they still need to simulate $n - k$ many Σ-protocol executions. Rather than requiring that all challenges sum to e, we will now require that they all lie on the same polynomial of degree $n - k$. The polynomial will be uniquely defined by the $n - k$ simulated and the received additional challenge. The challenges for the honest executions will be interpolated from this polynomial.

This approach requires us to program our NAPs at multiple points, but luckily such NAPs can easily be constructed from distributed point functions. Rather than viewing the NAP outputs as the evaluation of one function share, we can view them as the xor of the evaluation of multiple shares (for different point functions). The resulting key sizes would all increase by a multiplicative factor of k.

Dealing with Multiple Languages. When dealing with multiple languages, we may have to handle multiple response distributions, but we assume that all challenge sets are the same. Assume we have explainable samplers for all of these response distributions and assume that all of these samplers have the same randomness distribution $\mathcal{R}$. Then we can use a NAP with output distribution $\mathcal{R}$ to construct a NAP, where the output for each $j \in [n]$ comes from a different distribution.

4.3 Examples

Having established our generic transformation in the previous section, let us now look at two prominent examples of Σ-protocols that can be compiled with it. Without loss of generality, we

assume throughout this section that n and m are powers of two and hence their corresponding logarithm is a positive integer. Moreover, we use every explainable rejection sampler with precision λ, as detailed in Corollary 10.

$\mathsf{P}_1((G_0, G_1), \pi)$	$\mathsf{P}_2(\mathsf{aux}, e)$	$\mathsf{V}(a, e, z)$	$\mathsf{Sim}(e, z)$
1 : $\tau \leftarrow \mathcal{S}_m$	1 : **parse** aux as τ	1 : **if** $z(G_e) = a$:	1 : $a = z(G_e)$
2 : $G_2 := \tau(G_0)$	2 : $z_0 := \tau$	2 : **return** 1	2 : **return** a
3 : $a := G_2$	3 : $z_1 := \tau \circ \pi^{-1}$	3 : **else** :	
4 : $\mathsf{aux} = \tau$	4 : **return** $z := z_e$	4 : **return** 0	
5 : **return** (a, aux)			

Fig. 10. Σ-Protocol for the graph isomorphism problem by Goldreich, Micali, and Wigderson [GMW86].

Graph Isomorphism. One of the arguably most well-known Σ-protocols is that of Goldreich, Micali, and Wigderson [GMW86], which allows for showing that two graphs are isomorphic without revealing the secret isomorphism between them. More precisely, the statement is $x = (G_0, G_1)$, where G_0 and G_1 are two graphs each of which having m nodes, and the witness w is a permutation $\pi : [m] \to [m]$ which defines the isomorphism. We say $x \in \mathcal{L}^{\mathsf{GI}}$, if $\pi(G_0) = G_1$, where applying a permutation to a graph is interpreted as relabeling node i into node $\pi(i)$. We recall the protocol of Goldreich, Micali, and Wigderson in Fig. 10.

Besides being correct and special sound, we note that their protocol is also strong honest verifier zero-knowledge. To see this, we observe that the simulator, receiving a uniformly random challenge e and a uniformly random permutation $z : [m] \to [m]$, can deterministically compute $a = z(G_e)$. Furthermore, we observe that the challenge space is $\mathcal{E} = \{0, 1\}$ and the response distribution is the uniform distribution over the group of permutations $\mathcal{S}_m$. Using Theorem 5 in combination with our NAPs from Theorem 4, instantiated with the distributed point functions from Theorem 1, and using the explainable sampler from Corollary 13, we get the following theorem.

Theorem 6. *Let $\lambda, n, m \in \mathbb{N}$. Assuming the existence of collision-resistant hash functions, there exists a correct, computationally special sound, and honest verifier zero-knowledge Σ-protocol for the language $\mathcal{L}_{\mathsf{OR}}^{\mathsf{GI},n}$. The communication complexity of the protocol is $\mathcal{O}\big(m\lambda^3 \log n\big)$, where m denotes the number of nodes of the graphs in the statement.*

Let us explain how we arrive at the stated communication complexity. The transcript (a, e, z) of the constructed Σ-protocol is comprised of a hash $a \in \{0, 1\}^{\Theta(\lambda)}$, a challenge $e \in \mathcal{E} = \{0, 1\}$, and a response $z = (\mathsf{psk}_e, \mathsf{psk}_z)$, which is a pair of programmed NAP keys for input domain $\mathcal{X} = [n]$. To bound the bit length of the NAP keys, we use the bounds from Section 3.3. The bit length of psk_e for the output distribution $\mathcal{U}\{0, 1\}$ is in $\mathcal{O}(\lambda \log n)$. The bit length of psk_z is $\mathcal{O}(t\lambda \log n)$, where t is the number of randomness bits needed for the explainable sampler. To sample a uniformly random permutation from $\mathcal{S}_m$, we use the procedure described in Corollary 13 which requires rejection sampling of the uniform distribution over $[i]$ for $i \in \{2, \ldots, m\}$. We upper bound this by sampling m times over $[m]$. With λ bits of precision and constant M, we need at most $\lambda M(\log m + \lambda) \in \mathcal{O}\big(\lambda^2\big)$ many randomness bits for sampling over $[m]$. Here, we used that the number of nodes m is polynomial in λ. Hence, $t \in \mathcal{O}\big(m\lambda^2\big)$ and therefore the total bit length of psk_z is in $\mathcal{O}\big(m\lambda^3 \log n\big)$, which dominates the overall communication complexity.

Our theorem statement shows that one can prove the validity of one out of n instances of the graph isomorphism problem with a protocol that has a communication complexity that only depends logarithmically on n and that only relies on the existence of collision-resistant hashing.

$\mathsf{P}_1\,(G, \phi)$

1 : $\tau \leftarrow \mathcal{S}_3$
2 : **for** $\ell \in [m]$:
3 : $\quad r_\ell \leftarrow \{0,1\}^\lambda$
4 : $\quad c_\ell \leftarrow \mathsf{Commit}(\tau(\phi(\ell)); r_\ell)$
5 : $a := (c_1, \ldots, c_m)$
6 : $\mathsf{aux} = (\tau, r_1, \ldots, r_m)$
7 : **return** (a, aux)

$\mathsf{P}_2(\mathsf{aux}, e)$

1 : **parse** aux as $(\tau, r_1, \ldots, r_m)$
2 : **parse** e as (i, j)
3 : **for** $\ell \in [m] \setminus \{i, j\}$:
4 : $\quad c_\ell \leftarrow \mathsf{Commit}(\tau(\phi(\ell)); r_\ell)$
5 : $z := (r_i, r_j, \tau(\phi(i)), \tau(\phi(j)), (c_\ell)_{\ell \in [m] \setminus \{i,j\}})$
6 : **return** z

$\mathsf{V}(a, e, z)$

1 : **parse** a as $(c_1, \ldots, c_m)$
2 : **parse** e as (i, j)
3 : **parse** z as $(r_i, r_j, g_i, g_j, (c'_\ell)_{\ell \in [m] \setminus \{i,j\}})$
4 : $c'_i \leftarrow \mathsf{Commit}(g_i; r_i)$
5 : $c'_j \leftarrow \mathsf{Commit}(g_j; r_j)$
6 : **if** $g_i = g_j \vee (c_1, \ldots, c_m) \neq (c'_1, \ldots, c'_m)$:
7 : $\quad$ **return** 0
8 : **else** :
9 : $\quad$ **return** 1

$\mathsf{Sim}(e, z)$

1 : **parse** e as (i, j)
2 : **parse** z as $(r_i, r_j, g_i, g_j, (c_\ell)_{\ell \in [n] \setminus \{i,j\}})$
3 : $c_i \leftarrow \mathsf{Commit}(g_i; r_i)$
4 : $c_j \leftarrow \mathsf{Commit}(g_j; r_j)$
5 : $a = (c_1, \ldots, c_m)$
6 : **return** a

Fig. 11. A variant of the Σ-Protocol for graph 3-coloring by Goldreich, Micali, and Wigderson [GMW86].

Graph 3-Coloring. Another well-known Σ-protocol that was also presented by Goldreich, Micali, and Wigderson [GMW86] allows for showing that a given graph G is 3-colorable, i.e. that there exists a function ϕ, which assigns one of three colors to each node in a way that no two neighbours share a color. Here, $x = G$ is the statement and $w = \phi$ is the witness. More formally, let G be a graph with m nodes and E its set of edges. A 3-coloring of G is a function $\phi\colon [m] \to \{0,1,2\}$ such that for every edge $(i,j) \in E$ with $i, j \in [m]$, it yields $\phi(i) \neq \phi(j)$. We say $x \in \mathcal{L}^{\mathsf{G3C}}$, if there exists a 3-coloring ϕ of G. A slightly modified version of the original protocol is recalled in Figure 11. In the original protocol, the response z only contains the commitment openings of the two nodes specified by the challenge. The modification appends to the response z the commitments from the other edges (which can be recomputed with the help of the auxiliary information). This change is important to guarantee strong honest verifier zero-knowledge. The protocol involves a hiding and binding commitment scheme $\mathsf{Commit}\colon \{0,1,2\} \times \{0,1\}^\lambda \to \{0,1\}^{\Theta(\lambda)}$. In the following, we assume that the produced commitments of Commit are pseudorandom bit strings, i.e. computationally indistinguishable from uniformly random bit strings. Such commitment schemes can be build from one-way functions, as shown by Naor [Nao90].[5]

[5] The interactive commitment scheme originally presented in [Nao90] can be made non-interactive in the common reference string model by interpreting the verifier's (reusable) first message, which is a random bit string, as a crs.

In this setting, the protocol is not only correct and special sound, but also strong honest verifier zero-knowledge. To see the latter, we observe that the simulator, on input a uniformly random challenge e and a uniformly random response z, can deterministically compute the commitments for the i-th and j-th node (using the provided color and randomness in z) and output the full list of commitments (using the other commitments provided in z). The produced transcript is indistinguishable from a real one and due to the pseudorandomness of the commitment scheme, we can simulate those commitments by just picking uniformly random bit strings. The challenge space is given as $\mathcal{E} = E$. The response distribution is a product distribution of twice the uniform distribution over $\{0,1\}^\lambda$ (for the two commitment randomnesses), the uniform distribution over $[6]$ (for the two non-equal colors) and $n-2$ many uniform distributions over $\{0,1\}^{\Theta(\lambda)}$ (for the commitments). Overall, we obtain the following result.

Theorem 7. *Let $\lambda, n, m \in \mathbb{N}$. Assuming the existence of collision-resistant hash functions, there exists a correct, computationally special sound, and honest verifier zero-knowledge Σ-protocol for the language $\mathcal{L}_{\mathsf{OR}}^{\mathsf{G3C},n}$. The communication complexity of the protocol is $\mathcal{O}\big(m\lambda^3 \log n\big)$, where m denotes the number of nodes of the graphs in the statement.*

Let us again look at how we arrive at the stated communication complexity. Let (a, e, z) again be a transcript of the protocol. As before, the bit length of the hash a is $\mathcal{O}(\lambda)$, the bit length of e is $\log|E| \leq 2\log m$. Regarding the bit length of z, we observe that we need $t_1 = 2\lambda + (m-2)\mathcal{O}(\lambda) \in \mathcal{O}(m\lambda)$ many uniform output bits (covering the commitment randomness and the other commitments) and $t_2 = M\lambda(3+\lambda)$ many bits for the explainable sampler with constant M and λ bits of precision (for sampling over $[6]$). Thus, the total bit size of z is $\mathcal{O}\big(m\lambda^3 \cdot \log n\big)$, dominating the overall communication complexity.

5 Explainable Rejection Sampling

Previous works, such as that of Agrawal, Wichs, and Yamada [AWY20], showed that sampling (truncated) discrete gaussians via rejection sampling, as specified by Gentry, Peikert, and Vaikuntanathan [GPV08], is explainable[6]. Lu and Waters [LW22] formalized the concept of explainable samplers and show that large classes of distributions can be sampled in an explainable way. Here, we extend upon their result and show that anything that can be sampled efficiently via rejection sampling, can also be explained. Throughout this section, we assume that all involved distributions have efficiently computable probability density functions, i.e. that for any distribution $\mathcal{D}$ and any element x in $\mathsf{Supp}(\mathcal{D})$, we can compute the probability of x being sampled.

5.1 Textbook Rejection Sampling

Before presenting our new results, let us first recall textbook rejection sampling, depicted in Figure 12, along with the corresponding theorem statement in Theorem 8. For the sake of completeness, we provide a proof of this theorem in Section A.1. We note that this proof is not new to our work.

Theorem 8 (Rejection Sampling). *Let $\mathcal{P}, \mathcal{Q}$ be two discrete probability distributions such that $\mathsf{Supp}(\mathcal{P}) \subseteq \mathsf{Supp}(\mathcal{Q})$, where $\mathcal{Q}$ is the* starting *and $\mathcal{P}$ the* target distribution. *Further, let $M \in \mathbb{N}$ such that $\mathcal{P}(x)/\mathcal{Q}(x) \leq M$ for all $x \in \mathsf{Supp}(\mathcal{P})$. Then, the output of $\mathsf{RejSample}(\mathcal{P}, \mathcal{Q}, M)$ as defined in Figure 12 is distributed as $\mathcal{P}$. In expectation, the algorithm terminates after M trails.*

[6] In the work of Agrawal, Wichs, and Yamada [AWY20], the terminology *reversible* sampling was used.

$\mathsf{TextbookRejSample}(\mathcal{P}, \mathcal{Q}, M)$

1 : $\rho \leftarrow \mathcal{U}[0,1]$
2 : $x \leftarrow \mathcal{Q}$
3 : **if** $\rho \leq \mathcal{P}(x)/(M \cdot \mathcal{Q}(x))$:
4 : **return** x
5 : **else** :
6 : go to Step 1

Fig. 12. Textbook rejection sampling.

5.2 From Rejection Sampling to Explainable Samplers

Let us now see how rejection sampling can be used to construct explainable samplers. Recall, that rejection sampling repeatedly picks pairs (x, ρ), where $x \leftarrow \mathcal{Q}$ and $\rho \leftarrow \mathcal{U}[0,1]$, and then accepts x, if ρ is in some appropriate interval of $[0,1]$. As we have shown in the proof of Theorem 8, during each of these iterations, the algorithm will be terminating with probability $1/M$. Thus, the probability of not having terminated after $\lambda \cdot M$ iterations is bounded by

$$\left(1 - \frac{1}{M}\right)^{\lambda \cdot M} = \left(\left(1 - \frac{1}{M}\right)^{M}\right)^{\lambda} \leq e^{-\lambda}, \tag{1}$$

which is a negligible function in λ.

The explainable sampler we construct, outputs samples from some target distribution $\mathcal{P}$ and has randomness distribution $\mathcal{R} := (\mathcal{Q} \times \mathcal{U}[0,1])^{\lambda \cdot M}$ for some starting distribution $\mathcal{Q}$. In other words, our sampler takes $\kappa := \lambda \cdot M$ pairs as input, which are sufficient for simulating a real rejection sampling execution with overwhelming probability.

$\mathsf{RejSample}(1^\lambda, r)$

1 : parse r as $(x_i, \rho_i)_{i \in [\kappa]}$
2 : **for** i **in** $\{1, \ldots, \kappa\}$:
3 : **if** $\rho_i \leq \mathcal{P}(x_i)/(M \cdot \mathcal{Q}(x_i))$:
4 : **return** x_i
5 : **return** $\bot$

$\mathsf{RejExplain}(1^\lambda, x)$

1 : $r \leftarrow \mathcal{R}$
2 : parse r as $(x_i, \rho_i)_{i \in [\kappa]}$
3 : **for** i **in** $\{1, \ldots, \kappa\}$:
4 : **if** $\rho_i \leq \mathcal{P}(x_i)/(M \cdot \mathcal{Q}(x_i))$:
5 : $x_i := x$
6 : $\rho \leftarrow \mathcal{U}[0, \mathcal{P}(x_i)/(M \cdot \mathcal{Q}(x_i))]$
7 : $\rho_i := \rho$
8 : **return** r
9 : **return** $\bot$

Fig. 13. Explainable sampler via rejection sampling.

Theorem 9. *Let $\lambda \in \mathbb{N}$. Let $\mathcal{P} = \mathcal{P}(\lambda)$ and $\mathcal{Q} = \mathcal{Q}(\lambda)$ be two discrete probability distributions for which $\mathsf{Supp}(\mathcal{P}) \subseteq \mathsf{Supp}(\mathcal{Q})$ and for which there exists an $M = M(\lambda) \in \mathsf{poly}(\lambda)$, such that $\mathcal{P}(x)/\mathcal{Q}(x) \leq M$ for all $x \in \mathsf{Supp}(\mathcal{P})$. Then, the construction in Figure 13 defines a correct and explainable sampler for distribution $\mathcal{P}$ with randomness distribution $\mathcal{R}$, where $\mathcal{R} := (\mathcal{Q} \times \mathcal{U}[0,1])^{\lambda \cdot M}$.*

Proof. Let $\kappa := \lambda \cdot M$. Since both M and κ are polynomially bounded in λ, it directly follows that both $\mathsf{RejSample}$ and $\mathsf{RejExplain}$ are efficiently computable. Let us proceed to showing that our sampler is correct and explainable separately.

Correctness. To show correctness, we observe that for all $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A}$, it holds that

$$\begin{aligned}
&\left|\Pr\left[\mathcal{A}(x) = 1 : r \leftarrow \mathcal{R}, x \leftarrow \mathsf{RejSample}(1^\lambda; r)\right] - \Pr\left[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{P}\right]\right| \\
\leq &\Pr\left[x = \bot : r \leftarrow \mathcal{R}, x \leftarrow \mathsf{RejSample}(1^\lambda; r)\right] \\
+ &\left|\Pr\left[\mathcal{A}(x) = 1 : r \leftarrow \mathcal{R}, x \leftarrow \mathsf{RejSample}(1^\lambda; r) \mid x \neq \bot\right] - \Pr\left[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{P}\right]\right| \\
= &\,\mathsf{negl}(\lambda) + \left|\Pr\left[\mathcal{A}(x) = 1 : r \leftarrow \mathcal{R}, x \leftarrow \mathsf{RejSample}(1^\lambda; r) \mid x \neq \bot\right] - \Pr\left[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{P}\right]\right|,
\end{aligned}$$

since $\mathsf{RejSample}$ returns $\bot$ with negligible probability (cf. Equation 1). By Theorem 8, it holds that $\mathsf{RejSample}$ perfectly simulates sampling from $\mathcal{P}$, whenever it does not output $\bot$. Thus, it holds that

$$\left|\Pr\left[\mathcal{A}(x) = 1 : r \leftarrow \mathcal{R}, x \leftarrow \mathsf{RejSample}(1^\lambda; r) \mid x \neq \bot\right] - \Pr\left[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{P}\right]\right| = 0.$$

Therefore, the sampler is correct.

Explainability. Let $R \subset \mathsf{Supp}(\mathcal{R})$ be the set of random tapes on which $\mathsf{RejSample}$ successfully produces an output. That is, for all $r \in R$, it holds that $\mathsf{RejSample}(1^\lambda; r) \neq \bot$. Since, $\mathsf{RejSample}$ outputs $\bot$ with negligible probability, it follows that $|R| \geq (1 - \mathsf{negl}(\lambda)) \cdot |\mathsf{Supp}(\mathcal{P})|$, meaning that sampled random tapes from distribution $\mathcal{R}$ will be in R with overwhelming probability. Thus,

$$\begin{aligned}
&\left|\Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} r \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r) \end{matrix}\right] - \Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} r' \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r') \\ r \leftarrow \mathsf{RejExplain}(1^\lambda, x) \end{matrix}\right]\right| \\
\leq &\left|\Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} r \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r) \end{matrix} \,\middle|\, r \in R\right] - \Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} r' \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r') \\ r \leftarrow \mathsf{RejExplain}(1^\lambda, x) \end{matrix} \,\middle|\, r' \in R\right]\right| \\
&+ \mathsf{negl}(\lambda) \\
\leq &\left|\Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} r \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r) \end{matrix} \,\middle|\, r \in R\right] - \Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} x \leftarrow \mathcal{P} \\ r \leftarrow \mathsf{RejExplain}(1^\lambda, x) \end{matrix}\right]\right| \\
&+ \mathsf{negl}(\lambda),
\end{aligned}$$

where the last inequality follows from the correctness of our sampler.

For $r \in \mathsf{Supp}(\mathcal{R})$, let $\mathsf{Indx}(r) \in [\kappa] \cup \{\bot\}$ be the function that either outputs the first index $j \in [\kappa]$ that is accepted, i.e. such that $\rho_j \leq \mathcal{P}(x_j)/(M \cdot \mathcal{Q}(x_j))$, or, if no index is accepted, outputs $\bot$. By the definition of $\mathsf{RejExplain}$, we observe that

$$\begin{aligned}
&\Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} x \leftarrow \mathcal{P} \\ r \leftarrow \mathsf{RejExplain}(1^\lambda, x) \end{matrix}\right] \\
= &\Pr\left[\mathcal{A}(x,r) \colon \begin{matrix} r' := ((x_1, \rho_1), \ldots, (x_\kappa, \rho_\kappa)) \leftarrow \mathcal{R} \\ j = \mathsf{Indx}(r') \\ x \leftarrow \mathcal{P} \\ \rho \leftarrow \mathcal{U}[0, \mathcal{P}(x_j)/(M \cdot \mathcal{Q}(x_j))] \\ r := ((x_1, \rho_1), \ldots, (x_{j-1}, \rho_{j-1}), (x, \rho), (x_{j+1}, \rho_{j+1}), \ldots, (x_\kappa, \rho_\kappa)) \end{matrix} \,\middle|\, j \neq \bot\right] \pm \mathsf{negl}(\lambda).
\end{aligned}$$

Looking at the experiment in the last equation, we note that $\mathcal{R}$ is a product distribution and from the guarantees of rejection sampling it follows that x_j with $j = \mathsf{Indx}(r')$ is a sample from $\mathcal{P}$. The corresponding ρ_j is uniform conditioned on $\rho_j \leq \mathcal{P}(x_j)/(M \cdot \mathcal{Q}(x_j))$. Now, picking a fresh element $x \leftarrow \mathcal{P}$ along with $\rho \leftarrow \mathcal{U}[\mathcal{P}(x^*)/(M \cdot \mathcal{Q}(x^*))]$ is just the process of sampling a new pair from the same distribution as that of (x_j, ρ_j). Replacing one pair by the other does not affect the output distribution and therefore

$$\left| \Pr\left[\mathcal{A}(x,r) \colon \begin{array}{r} r \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r) \end{array}\right] - \Pr\left[\mathcal{A}(x,r) \colon \begin{array}{r} r' \leftarrow \mathcal{R} \\ x \leftarrow \mathsf{RejSample}(1^\lambda; r') \\ r \leftarrow \mathsf{RejExplain}(1^\lambda, x) \end{array}\right] \right| \leq \mathsf{negl}(\lambda),$$

which shows explainability. □

5.3 Handling Finite Precision

In the above, we assumed that we can sample from the continuous distribution $\mathcal{U}[0,1]$, but in reality[7] we can clearly only sample from a discrete distribution. Let us note that sampling uniform p-bit integers allows for simulating rejection sampling and our explainable sampler with an additive error of $\mathcal{O}(2^{-p})$. We will not provide a formal proof here, but still provide an intuition of how the statement can be proven easily.

To see that the above claim is true, note that sampling ρ from $\mathcal{U}[0,1]$ and checking $\rho \leq t$ for threshold $t \in [0,1]$, where $1/t$ divides 2^p, can be perfectly simulated by sampling ρ' from $\{0, \ldots, 2^p - 1\}$ and checking whether $\rho' \leq t \cdot 2^p$. Furthermore, note that any arbitrary threshold $t \in [0,1]$ is at most an additive factor away from a threshold t' that divides 2^p. The outcome between using the two thresholds only differs for $\rho \in [t, t']$, which happens with probability $\mathcal{O}(2^{-p})$.

Corollary 10. *Let $\lambda, p \in \mathbb{N}$ with $p = \Omega(\lambda)$. Let $\mathcal{P} = \mathcal{P}(\lambda)$ and $\mathcal{Q} = \mathcal{Q}(\lambda)$ be two discrete probability distributions for which $\mathsf{Supp}(\mathcal{P}) \subseteq \mathsf{Supp}(\mathcal{Q})$ and for which there exists an $M = M(\lambda) \in \mathsf{poly}(\lambda)$, such that $\mathcal{P}(x)/\mathcal{Q}(x) \leq M$ for all $x \in \mathsf{Supp}(\mathcal{P})$. Then, there exists a correct and explainable sampler for distribution $\mathcal{P}$ with randomness distribution $\mathcal{R}$, where $\mathcal{R} := (\mathcal{Q} \times \mathcal{U}\{0, \ldots, 2^p - 1\})^{\lambda \cdot M}$.*

5.4 Uniform Distributions Between Sets of Different Sizes

A simple, but useful case of rejection sampling is to use the uniform distribution $\mathcal{Q}$ over random bit strings to sample from some uniform distribution $\mathcal{P}$, where $|\mathsf{Supp}(\mathcal{P})|$ is not a power of two. Setting $M = |\mathsf{Supp}(\mathcal{Q})|/|\mathsf{Supp}(\mathcal{P})|$, we get that $\mathcal{P}(x)/(M \cdot \mathcal{Q}(x))$ is one if $x \in \mathsf{Supp}(\mathcal{P})$, and zero otherwise. In simpler words, the rejection sampler outputs the first element sampled from $\mathcal{Q}$, which also lies in $\mathcal{P}$.

Corollary 11. *Let $\lambda, p \in \mathbb{N}$ with $p = \Omega(\lambda)$. Let $\mathcal{P} = \mathcal{P}(\lambda)$ and $\mathcal{Q} = \mathcal{Q}(\lambda)$ be two uniform, discrete probability distributions with $\mathsf{Supp}(\mathcal{P}) \subseteq \mathsf{Supp}(\mathcal{Q})$ and $|\mathsf{Supp}(\mathcal{Q})|/|\mathsf{Supp}(\mathcal{P})| = \mathsf{poly}(\lambda)$. Then there exists a correct and explainable sampler with $M = |\mathsf{Supp}(\mathcal{Q})|/|\mathsf{Supp}(\mathcal{P})|$ for distribution $\mathcal{P}$ and randomness distribution $\mathcal{R} := (\mathcal{Q} \times \mathcal{U}\{0, \ldots, 2^p - 1\})^{\lambda \cdot M}$.*

[7] In particular, this is the case in our NAP constructions from Section 3.

5.5 Explainable Samplers for Product Distributions and Permutations

Given explainable samplers for distributions $\mathcal{P}_1, \ldots, \mathcal{P}_m$ for $m \in \mathsf{poly}(\lambda)$ with randomness distributions $\mathcal{R}_1, \ldots, \mathcal{R}_m$ respectively, one can easily construct an explainable sampler for the product distribution $\mathcal{P}_1 \times \cdots \times \mathcal{P}_m$ with randomness domain $\mathcal{R}_1 \times \cdots \times \mathcal{R}_m$ by simply running all individual explainable samplers in parallel. Correctness and explainability of this sampler follows via a standard hybrid argument.

Corollary 12. *Let $\lambda, m \in \mathbb{N}$ with $m = \mathsf{poly}(\lambda)$. For $i \in [m]$, let ES_i be a correct and explainable sampler for distribution $\mathcal{P}_i = \mathcal{P}_i(\lambda)$ with randomness distribution $\mathcal{R}_i = \mathcal{R}_i(\lambda)$. Then there exists a correct and explainable sampler for distribution $\mathcal{P}_1 \times \cdots \times \mathcal{P}_m$ with randomness distribution $\mathcal{R}_1 \times \cdots \times \mathcal{R}_m$.*

This corollary is particularly useful, as it allows us to construct an explainable sampler for the uniform distribution over $\mathcal{S}_m$, i.e., uniformly random permutations over $[m]$. To see how, let us first recall the Fisher-Yates shuffle, which takes a list of input values $(a_1, \ldots, a_m)$ as input and returns a uniformly random permutation $(b_1, \ldots, b_m)$ thereof. The shuffle initializes a set $A = \{a_1, \ldots, a_m\}$ and a counter $c = 1$. It then repeatedly picks a uniformly random element $a \in A$, assigns $b_c := a$, removes a from A, and increments c by one until $A = \emptyset$. In other words, in the first step, it selects a uniformly random index $i \in [m]$ and assigns $b_1 := a_i$, then it selects a uniformly random index j in $[m-1]$ and assigns b_2 to be the j-th elements among the remaining ones and so on.

From the above, we can see that any permutation over $[m]$, corresponds to exactly one element from the set $[m] \times \cdots \times [2]$. Thus, the task of sampling a uniformly random permutation is identical to the task of sampling a uniformly random element in $[m] \times \cdots \times [2]$. Using Corollary 12, we can then obtain an explainable sampler for permutations from explainable samplers for uniform distributions over the sets $[i]$ for $i \in [m]$.

Corollary 13. *Let $\lambda, m \in \mathbb{N}$ with $m = \mathsf{poly}(\lambda)$. For $i \in [m] \setminus \{1\}$, let ES_i be a correct and explainable sampler for the uniform distribution over $[i]$ with randomness distribution $\mathcal{R}_i = \mathcal{R}_i(\lambda)$. Then there exists a correct and explainable sampler for the uniform distribution over the set $\mathcal{S}_m$ of all permutations $\pi : [m] \to [m]$ with randomness distribution $\mathcal{R}_2 \times \cdots \times \mathcal{R}_m$.*

Acknowledgement

We thank Rafail Ostrovsky for pointing us to the works of De Santis et al. [DDP+94] and Hemenway et al. [HJO+16], which we had missed.

References

[AC20] Thomas Attema and Ronald Cramer. "Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 513–543. DOI: 10.1007/978-3-030-56877-1_18.

[ACF21] Thomas Attema, Ronald Cramer, and Serge Fehr. "Compressing Proofs of k-Out-Of-n Partial Knowledge". In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 65–91. DOI: 10.1007/978-3-030-84259-8_3.

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. "A Compressed Σ-Protocol Theory for Lattices". In: *CRYPTO 2021, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 549–579. DOI: 10.1007/978-3-030-84245-1_19.

[AS92] Sanjeev Arora and Shmuel Safra. "Probabilistic Checking of Proofs; A New Characterization of NP". In: *33rd FOCS*. IEEE Computer Society Press, Oct. 1992, pp. 2–13. DOI: 10.1109/SFCS.1992.267824.

[AWY20] Shweta Agrawal, Daniel Wichs, and Shota Yamada. "Optimal Broadcast Encryption from LWE and Pairings in the Standard Model". In: *TCC 2020, Part I*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12550. LNCS. Springer, Heidelberg, Nov. 2020, pp. 149–178. DOI: 10.1007/978-3-030-64375-1_6.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[BBH+18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046. 2018.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 327–357. DOI: 10.1007/978-3-662-49896-5_12.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *TCC 2016-B, Part II*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9986. LNCS. Springer, Heidelberg, 2016, pp. 31–60. DOI: 10.1007/978-3-662-53644-5_2.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 337–367. DOI: 10.1007/978-3-662-46803-6_12.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. "Constraining Pseudorandom Functions Privately". In: *PKC 2017, Part II*. Ed. by Serge Fehr. Vol. 10175. LNCS. Springer, Heidelberg, Mar. 2017, pp. 494–524. DOI: 10.1007/978-3-662-54388-7_17.

[BR08] Mihir Bellare and Todor Ristov. "Hash Functions from Sigma Protocols and Improvements to VSH". In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 125–142. DOI: 10.1007/978-3-540-89255-7_9.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols". In: *CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 174–187. DOI: 10.1007/3-540-48658-5_19.

[DDP+94] Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano, and Moti Yung. "On Monotone Formula Closure of SZK". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 454–465. DOI: 10.1109/SFCS.1994.365745.

[GGHA+22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. "Stacking Sigmas: A Framework to Compose Σ-Protocols for Disjunctions". In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski.

Vol. 13276. LNCS. Springer, Heidelberg, 2022, pp. 458–487. DOI: 10.1007/978-3-031-07085-3_16.

[GI14] Niv Gilboa and Yuval Ishai. "Distributed Point Functions and Their Applications". In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 640–658. DOI: 10.1007/978-3-642-55220-5_35.

[GK15] Jens Groth and Markulf Kohlweiss. "One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 253–280. DOI: 10.1007/978-3-662-46803-6_9.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design (Extended Abstract)". In: *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 174–187. DOI: 10.1109/SFCS.1986.47.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206. DOI: 10.1145/1374376.1374407.

[HIL+99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. "A Pseudorandom Generator from any One-way Function". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. "Adaptively Secure Garbled Circuits from One-Way Functions". In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 149–178. DOI: 10.1007/978-3-662-53015-3_6.

[IKO+09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Zero-Knowledge Proofs from Secure Multiparty Computation". In: *SIAM J. Comput.* 39.3 (2009), pp. 1121–1152.

[Kil92] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *24th ACM STOC*. ACM Press, May 1992, pp. 723–732. DOI: 10.1145/129712.129782.

[LW22] George Lu and Brent Waters. "How to Sample a Discrete Gaussian (and more) from a Random Oracle". In: *TCC 2022, Part II*. LNCS. Springer, Heidelberg, Nov. 2022, pp. 653–682. DOI: 10.1007/978-3-031-22365-5_23.

[Nao90] Moni Naor. "Bit Commitment Using Pseudo-Randomness". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 128–136. DOI: 10.1007/0-387-34805-0_13.

[PS18] Chris Peikert and Sina Shiehian. "Privately Constraining and Programming PRFs, the LWE Way". In: *PKC 2018, Part II*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. LNCS. Springer, Heidelberg, Mar. 2018, pp. 675–701. DOI: 10.1007/978-3-319-76581-5_23.

[PS20] Chris Peikert and Sina Shiehian. "Constraining and Watermarking PRFs from Milder Assumptions". In: *PKC 2020, Part I*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12110. LNCS. Springer, Heidelberg, May 2020, pp. 431–461. DOI: 10.1007/978-3-030-45374-9_15.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-round interactive proofs for delegating computation". In: *48th ACM STOC*. Ed. by

Daniel Wichs and Yishay Mansour. ACM Press, June 2016, pp. 49–62. DOI: 10.1145/2897518.2897652.

[WW22] Brent Waters and David J. Wu. "Batch Arguments for NP and More from Standard Bilinear Group Assumptions". In: *CRYPTO 2022, Part II*. LNCS. Springer, Heidelberg, Aug. 2022, pp. 433–463. DOI: 10.1007/978-3-031-15979-4_15.

Appendix A Missing Proofs

In this section we provide proofs (or proof sketches) for all theorem statements from the main body that were missing a proof.

A.1 Rejection Sampling Produces the Correct Distribution (Proof of Theorem 8)

To prove the theorem statement, let us show that.

$$\Pr[x \leftarrow \mathsf{TextbookRejSample}(\mathcal{P}, \mathcal{Q}, M)] = \mathcal{P}(x).$$

Towards this goal, let us define some events. We will abuse notation and write $x \leftarrow \mathcal{Q}$ to denote the event that $\mathcal{Q}$ is sampled and the obtained value is x. Let $\mathsf{reject}(x)$ be the event that a fixed $x \in \mathsf{Supp}(\mathcal{Q})$ is rejected, i.e. that for a uniformly random $\rho \in \mathcal{U}[0,1]$, it holds that $\rho > \mathcal{P}(x)/(M \cdot \mathcal{Q}(x))$. Let $\mathsf{accept}(x)$ be the event that a fixed $x \in \mathsf{Supp}(\mathcal{Q})$ is not rejected, i.e. $\Pr[\mathsf{reject}(x)] = 1 - \Pr[\mathsf{accept}(x)]$ for all $x \in \mathsf{Supp}(\mathcal{Q})$. Let reject and accept be the events that a value x is sampled according to distribution $\mathcal{Q}$ and then rejected or accepted respectively. Let us start by observing that

$$\begin{aligned}\Pr[x \leftarrow \mathsf{TextbookRejSample}(\mathcal{P}, \mathcal{Q}, M)] &= \sum_{i=1}^{\infty} \Pr[\mathsf{reject} \text{ happens } i-1 \text{ times} \wedge x \leftarrow \mathcal{Q} \wedge \mathsf{accept}(x)] \\ &= \sum_{i=1}^{\infty} \Pr[\mathsf{reject}]^{i-1} \cdot \Pr[x \leftarrow \mathcal{Q} \wedge \mathsf{accept}(x)],\end{aligned}$$

where the last equality follows from the fact that each iteration through steps 1 to step 3 is independent of each other.

$$\begin{aligned}\Pr[x \leftarrow \mathcal{Q} \wedge \mathsf{accept}(x)] &= \Pr[x \leftarrow \mathcal{Q}] \cdot \Pr[\mathsf{accept}(x) \mid x \leftarrow \mathcal{Q}] \\ &= \mathcal{Q}(x) \cdot \Pr\left[\rho \leq \frac{\mathcal{P}(x)}{M \cdot \mathcal{Q}(x)} : \rho \leftarrow \mathcal{U}[0,1] \;\middle|\; x \leftarrow \mathcal{Q}\right] \\ &= \mathcal{Q}(x) \cdot \frac{\mathcal{P}(x)}{M \cdot \mathcal{Q}(x)} = \frac{\mathcal{P}(x)}{M}.\end{aligned}$$

Furthermore, we also observe that

$$\begin{aligned}\Pr[\mathsf{reject}] &= \sum_{y \in \mathsf{Supp}(\mathcal{Q})} \Pr[y \leftarrow \mathcal{Q} \wedge \mathsf{reject}(y)] \\ &= \sum_{y \in \mathsf{Supp}(\mathcal{Q})} \Pr[y \leftarrow \mathcal{Q}] \cdot \Pr[\mathsf{reject}(y) \mid y \leftarrow \mathcal{Q}] \\ &= \sum_{y \in \mathsf{Supp}(\mathcal{Q})} \mathcal{Q}(y) \cdot \Pr\left[\rho > \frac{\mathcal{P}(y)}{M \cdot \mathcal{Q}(y)} : \rho \leftarrow \mathcal{U}[0,1] \middle| y \leftarrow \mathcal{Q}\right] \\ &= \sum_{y \in \mathsf{Supp}(\mathcal{Q})} \mathcal{Q}(y) \cdot \left(1 - \frac{\mathcal{P}(y)}{M \cdot \mathcal{Q}(y)}\right)\end{aligned}$$

$$= \sum_{y \in \mathsf{Supp}(\mathcal{Q})} \left(\mathcal{Q}(y) - \frac{\mathcal{P}(y)}{M} \right) = 1 - \frac{1}{M},$$

where in the last equation we used the fact that $\mathsf{Supp}(\mathcal{P}) \subseteq \mathsf{Supp}(\mathcal{Q})$. Putting both terms together, we obtain

$$\begin{aligned}
\Pr[x \leftarrow \mathsf{TextbookRejSample}(\mathcal{P}, \mathcal{Q}, M)] &= \sum_{i=1}^{\infty} \Pr[\mathsf{reject}]^{i-1} \cdot \Pr[x \leftarrow \mathcal{Q} \wedge \mathsf{accept}(x)] \\
&= \sum_{i=1}^{\infty} \left(1 - \frac{1}{M}\right)^{i-1} \cdot \frac{\mathcal{P}(x)}{M} \\
&= \mathcal{P}(x) \sum_{i=1}^{\infty} \left(1 - \frac{1}{M}\right)^{i-1} \cdot \frac{1}{M} \\
&= \mathcal{P}(x),
\end{aligned}$$

where the last equation is obtained by observing that the last sum is a geometric series.

Lastly, let us argue that the rejection sampling procedure terminates after M trials in expectations. To see this, we note that by the above calculations, it holds that $\Pr[\mathsf{accept}] = 1/M$. Since the individual iterations of step 1 to step 3 are independent of each other, it follows that the running time of $\mathsf{RejSample}(\mathcal{P}, \mathcal{Q}, M)$ follows a geometric distribution and thus the theorem statement follows. □

A.2 From Single Bit NAPs to Multiple Bit NAPs (Proof of Theorem 3)

$\mathsf{Gen}(1^\lambda, x^*)$

1 : **for** $i \in [t]$:
2 : $\quad \mathsf{msk}_i \leftarrow \mathsf{NAP}'.\mathsf{Gen}(1^\lambda, x^*)$
3 : **return** $(\mathsf{msk}_1, \ldots, \mathsf{msk}_t)$

$\mathsf{Eval}(\mathsf{msk}, x)$

1 : parse msk as $(\mathsf{msk}_1, \ldots, \mathsf{msk}_t)$
2 : **for** $i \in [t]$:
3 : $\quad y_i \leftarrow \mathsf{NAP}'.\mathsf{Eval}(\mathsf{msk}_i, x)$
4 : **return** $y_1 \| \ldots \| y_t$

$\mathsf{Prog}(\mathsf{msk}, y^*)$

1 : parse y^* as $(y_1^*, \ldots, y_t^*)$
2 : parse msk as $(\mathsf{msk}_1, \ldots, \mathsf{msk}_t)$
3 : **for** $i \in [t]$:
4 : $\quad \mathsf{psk}_i \leftarrow \mathsf{NAP}'.\mathsf{Prog}(\mathsf{msk}_i, y_i^*)$
5 : **return** $(\mathsf{psk}_1, \ldots, \mathsf{psk}_t)$

$\mathsf{PEval}(\mathsf{psk}, x)$

1 : parse psk as $(\mathsf{psk}_1, \ldots, \mathsf{psk}_t)$
2 : **for** $i \in [t]$:
3 : $\quad y_i \leftarrow \mathsf{NAP}'.\mathsf{PEval}(\mathsf{psk}_i, x)$
4 : **return** $y_1 \| \ldots \| y_t$

Fig. 14. NAP for uniform output distribution and t-bit outputs.

To prove the theorem statement, we need to provide the desired construction and show that it is both correct and privately programmable. The construction is depicted in Figure 14. In the following, let us sketch the proof of each property:

Correctness. Directly follows by inspection of the construction.

Private Programmability. Since NAP' is privately programmable by assumption, there exists a simulator $\mathsf{NAP}'.\mathsf{Sim}$ that outputs a simulated psk on input 1^λ. We construct a simulator Sim by calling the simulator $\mathsf{NAP}'.\mathsf{Sim}$ with fresh random coins t times and concatenating the outputs, i.e. by computing $\mathsf{psk}_i \leftarrow \mathsf{NAP}'.\mathsf{Sim}(1^\lambda)$ for $i \in [t]$ and returning $(\mathsf{psk}_1, \ldots, \mathsf{psk}_t)$.

For $i \in [t]$, let the function $\mathsf{Prog}_i(\mathsf{msk}, y^*)$ be the function which takes $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_t)$ and $y^* \in \{0,1\}^t$ as input and returns a key $\mathsf{psk} = (\mathsf{psk}_1, \ldots, \mathsf{psk}_t)$, where for all $j \in [t]$ with $j \leq i$, we compute $\mathsf{psk}_j \leftarrow \mathsf{Sim}(1^\lambda)$ and for all $j > i$, we compute $\mathsf{psk}_j \leftarrow \mathsf{NAP}'.\mathsf{Prog}(\mathsf{msk}_j, y^*_j)$. Let Hybrid_0 be identical to the experiment $\mathsf{Expt}^{\mathsf{RealPP}}_{\mathcal{A}}(1^\lambda)$ and for $i \in [t]$, let Hybrid_i be identical to Hybrid_0 with the difference being that psk is computed via $\mathsf{NAP}.\mathsf{Prog}_i$. We note that Hybrid_t is identical to $\mathsf{Expt}^{\mathsf{IdealPP}}_{\mathcal{A}}(1^\lambda)$, when the simulator Sim we described above is used. From here, private programmability of the construction in Figure 14 follows via a standard hybrid argument and the private programmability of NAP'. □

A.3 Building NAPs with Different Output Distributions (Proof of Theorem 4)

To prove the theorem statement, we consider each of the properties of a NAP individually:

Correctness. To show correctness, we need to show that for all $\lambda \in \mathbb{N}$ and all $x^* \in \mathcal{X}$, there exists a negligible function $\mathsf{negl}(\lambda)$, such that it simultaneously holds that

$$\Pr\left[\begin{array}{l}\exists x \in \mathcal{X} \setminus \{x^*\} \\ \mathsf{PEval}(\mathsf{psk}, x) \neq \mathsf{Eval}(\mathsf{msk}, x)\end{array} : \begin{array}{r}\mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ y^* \leftarrow \mathcal{R} \\ \mathsf{psk} \leftarrow \mathsf{Prog}(\mathsf{msk}, y^*)\end{array}\right] \leq \mathsf{negl}(\lambda),$$

and

$$\Pr\left[\mathsf{PEval}(\mathsf{psk}, x^*) \neq y^* \quad : \begin{array}{r}\mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ y^* \leftarrow \mathcal{R} \\ \mathsf{psk} \leftarrow \mathsf{Prog}(\mathsf{msk}, y^*)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

We observe that the first of those two inequalities is only concerned with the points that are not programmed. Since

$$\mathsf{PEval}(\mathsf{psk}, x) = \mathsf{ES}.\mathsf{Sample}(1^\lambda, \mathsf{NAP}'.\mathsf{PEval}(\mathsf{psk}, x)),$$

and since $\mathsf{ES}.\mathsf{Sample}$ is just a deterministic function of its given random coins, it directly follows from the correctness of NAP' that the first inequality is satisfied.

Regarding the second inequality, let us start by observing that ES is a correct and explainable sampler. From there it follows that

$$\Pr\left[\mathsf{ES}.\mathsf{Sample}(1^\lambda, r) \neq x \quad : \begin{array}{r}x \leftarrow \mathcal{D} \\ r \leftarrow \mathsf{ES}.\mathsf{Explain}(1^\lambda, x)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

There are exactly two ways in which correctness on the programmed point can fail. Either because NAP' or because ES behaved incorrectly. More concretely, by a union bound, we observe that

$$\Pr\left[\mathsf{PEval}(\mathsf{psk}, x^*) \neq y^* \quad : \begin{array}{r}\mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ y^* \leftarrow \mathcal{R} \\ \mathsf{psk} \leftarrow \mathsf{Prog}(\mathsf{msk}, y^*)\end{array}\right]$$

$$
\begin{aligned}
&= \Pr\left[\mathsf{PEval}(\mathsf{psk}, x^*) \neq y^* \quad : \begin{array}{r} \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ y^* \leftarrow \mathcal{R} \\ r^* \leftarrow \mathsf{ES.Explain}(1^\lambda, y^*) \\ \mathsf{psk} \leftarrow \mathsf{NAP}'.\mathsf{Prog}(\mathsf{msk}, r^*) \end{array}\right] \\
&\leq \Pr\left[\mathsf{ES.Sample}(1^\lambda, r^*) \neq y^* \quad : \begin{array}{r} \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ y^* \leftarrow \mathcal{R} \\ r^* \leftarrow \mathsf{ES.Explain}(1^\lambda, y^*) \\ \mathsf{psk} \leftarrow \mathsf{NAP}'.\mathsf{Prog}(\mathsf{msk}, r^*) \end{array}\right] \\
&+ \Pr\left[\mathsf{NAP}'.\mathsf{PEval}(\mathsf{psk}, x^*) \neq r^* \quad : \begin{array}{r} \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda, x^*) \\ y^* \leftarrow \mathcal{R} \\ r^* \leftarrow \mathsf{ES.Explain}(1^\lambda, y^*) \\ \mathsf{psk} \leftarrow \mathsf{NAP}'.\mathsf{Prog}(\mathsf{msk}, r^*) \end{array}\right] \leq \mathsf{negl}(\lambda).
\end{aligned}
$$

and thus our construction satisfies correctness.

Private Programmability. By assumption, NAP' is privately programmable and therefore there exists a PPT simulator Sim for which it holds that no PPT adversary can distinguish the real and the ideal private programmability experiment with non-negligible advantage. We define Sim to be the simulator for NAP and claim that the same is true for the private programmability experiment with respect to NAP.

Let Hybrid_0 be the private programmability experiments against NAP. Let Hybrid_1 be identical to Hybrid_0, apart from step 2 in $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealPP}}(1^\lambda)$, where instead of computing $y^* \leftarrow \mathcal{D}$, we will now sample $\tilde{r} \leftarrow \mathcal{R}$ and compute $y^* \leftarrow \mathsf{ES.Sample}(1^\lambda, \tilde{r})$. Indistinguishability of hybrids Hybrid_0 and Hybrid_1 follows from the correctness of the explainable sampler.

Let Hybrid_2 be identical to Hybrid_1, apart from step 4 in the modified $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{RealPP}}(1^\lambda)$ from Hybrid_1, where during the computation of psk we will no longer compute $r^* \leftarrow \mathsf{Explain}(1^\lambda, y^*)$, but instead set $r^* := \tilde{r}$. Note that at this point, the value y^* is not used anymore in the experiment. Indistinguishability of hybrids Hybrid_1 and Hybrid_2 follows from the explainability of ES.

At this point we observe that the experiments in Hybrid_2 are identical to the experiments against NAP' with simulator Sim. Since NAP' is privately programmable, it follows that so is NAP. □