Towards a Unified Approach to Black-Box Constructions of Zero-Knowledge Proofs*

Xiao Liang and Omkant Pandey

Stony Brook University, Stony Brook NY 11790, USA
{liang1,omkant}@cs.stonybrook.edu

Abstract. General-purpose zero-knowledge proofs for all NP languages greatly simplify secure protocol design. However, they inherently require the code of the underlying relation. If the relation contains black-box calls to a cryptographic function, the code of that function must be known to use the ZK proof, even if both the relation and the proof require only black-box access to the function. Rosulek (Crypto'12) shows that non-trivial proofs for even simple statements, such as membership in the range of a one-way function, require non-black-box access.

We propose an alternative approach to bypass Rosulek's impossibility result. Instead of asking for a ZK proof directly for the given one-way function f, we seek to construct a *new* one-way function F given only black-box access to f, *and* an associated ZK protocol for proving non-trivial statements, such as range membership, over its output. We say that F, along with its proof system, is a *proof-based* one-way function. We similarly define proof-based versions of other primitives, specifically pseudo-random generators and collision-resistant hash functions.

We show how to construct proof-based versions of each of the primitives mentioned above from their ordinary counterparts under mild but necessary restrictions over the input. More specifically,

- We first show that if the prover entirely chooses the input, then proof-based pseudo-random generators cannot be constructed from ordinary ones in a black-box manner, thus establishing that some restrictions over the input are necessary.
- We next present black-box constructions handling inputs of the form (x, r) where r is chosen uniformly by the verifier. This is similar to the restrictions in the widely used Goldreich-Levin theorem. The associated ZK proofs support range membership over the output as well as arbitrary predicates over prefixes of the input.

Our results open up the possibility that general-purpose ZK proofs for relations that require black-box access to the primitives above may be possible in the future without violating their black-box nature by instantiating them using proof-based primitives instead of ordinary ones.

Keywords: Zero-Knowledge · Black-Box · Separation

* This material is based upon work supported in part by DARPA SIEVE Award HR00112020026, NSF grants 1907908 and 2028920, and a Cisco Research Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government, DARPA, NSF, or Cisco.

Table of Contents

1 Introduction

Zero-knowledge proofs (ZKPs) are a method to prove that a statement is true without revealing any additional knowledge [GMR85]. A significant achievement in cryptography has been the construction of ZKPs for NP-complete problems [GMW86]. Since every NP relation can be efficiently reduced to any NP-complete relation [Coo71, Kar72, Lev73], this yields a ZKP for all languages in NP. Due to this reason, ZKPs for NP-complete problems are often called *general-purpose* proofs. As evidenced by numerous follow-up works, general-purpose proofs have been incredibly useful to the theory of cryptography.

Early constructions of general-purpose ZKPs required only *black-box* access to any one-way function (OWF), i.e., they used the given OWF as an *oracle*. A *black-box construction* of this kind thus depends only on the input/output behavior of the given cryptographic primitive. In particular, it is independent of the specific implementation or *code* of the primitive.

A black-box construction is often preferred over a non-black one due to its attractive properties. For example, it remains valid even if the primitive/oracle is based on a *physical* object such as a noisy-channel or tamper-proof hardware [Wyn75, CK88, GLM+04]. Also, its efficiency does not depend on the implementation details of the primitive, thus establishing that efficiency can be theoretically independent of the primitive's code.

Unfortunately, general-purpose proofs are unsuitable when seeking a black-box construction for some desired cryptographic task since they inherently require the full code of the underlying relation to perform the NP reduction. In other words, if the relation requires black-box access to a OWF, the code of the OWF must be known *even though* neither the ZKP nor the relation needs it. In fact, this has been the main reason for the non-black-box nature of many cryptographic constructions that are otherwise optimal. Analogous black-box constructions often require significant effort and technical innovation, as evidenced by the secure computation literature, e.g., [Kil88, DI05, IKLP06, IKOS07, Hai08, IPS08, PW09, Wee10, Goy11, GLOV12, LP12, Kiy14, GGMP16, HV16, GKP18, CLP20, GLPV20].

In light of the above situation, it is tempting to imagine a "dream version" of general-purpose proofs where, if the underlying relation $\mathcal{R}$ requires black-box access to a cryptographic function f, say from a specified class such as the class of OWFs, then so should the general-purpose ZKP for proving membership in $\mathcal{R}$. We informally refer to such relations as *black-box relations*. Such a result, if possible, would greatly simplify the task of future black-box constructions and potentially unify the diverse set of techniques that exist in this area.

As one might suspect, this dream version is too good to be true. In his beautiful work, Rosulek [Ros12] rules out ZKPs for proving *membership in the range* of a OWF f given as an oracle. More specifically, assuming injective OWFs, Rosulek rules out (even honest-verifier) *witness-hiding* protocols [FS90] for the relation $\mathcal{R}^f = \{(y, x) \mid y = f(x)\}$ where f is chosen from the class of all OWFs and provided as an oracle to the protocol.

In contrast to the negative result for OWFs, a large body of literature constructs so-called *black-box commit-and-prove* protocols [IKOS07, GLOV12, GOSV14, HV18, KOS18, Kiy20]. Informally speaking, a commit-and-prove protocol between a committer and a receiver ensures that at the end of the protocol, the committer is committed to some hidden value satisfying a predefined property. This primitive can be constructed with only black-box access to an ordinary commitment scheme which may originally not support any proofs whatsoever. In many situations, commit-and-prove protocols serve as a good substitute for ordinary commitments; moreover, their ability to support proofs over committed values makes them a great tool for constructing larger black-box protocols.

In hindsight, we can view black-box commit-and-prove protocols as an alternative to bypass the aforementioned negative result of [Ros12]. That is, instead of constructing ZKP directly for every OWF, we ask the following indirect question:

> *Given only black-box access to an OWF f, can we construct a new OWF F and a ZKP system Π_F for proving membership in the range of F?*

Of course, we can ask for general properties instead of merely range membership.

The idea is that F can be used as a substitute for f in any computation $C^{(\cdot)}$ that requires only black-box access to OWFs. More importantly, it gives hope that general-purpose black-box ZKPs for proving the correctness of computation $C^{(\cdot)}$ may be possible since the correctness of responses from F can be ensured using Π_F, all while requiring only black-box access to f. We remark that we do not obtain such a result for general computations in this work and merely point out that the existence of (F, Π_F) may open a path towards it.

We call the pair (F, Π_F) a *proof-based one-way function* (PB-OWF). Analogously, we consider proof-based versions of other primitives, specifically pseudo-random generators (PRGs) and collision-resistant hash functions (CRHFs). Motivated by the aforementioned possibility of a general-purpose proof system for black-box cryptographic computations $C^{(\cdot)}$, this paper initiates a study of black-box constructions of proof-based cryptographic primitives. We obtain a mix of both negative and positive results as outlined below.

1.1 Our Results

Given the existence of black-box commit-and-prove protocols, it is not unreasonable to expect that black-box proof-based versions of OWFs, PRGs, and CRHFs may also exist. Interestingly, the fact that these primitives are *deterministic* functions truly separates them from commitments. The existence of their proof-based versions seems to depend on how we view the input, as discussed below.

Negative Results via Black-Box Separation. In common applications of non-interactive primitives such as OWFs and PRGs, the entire input is usually controlled by the evaluator of these functions. We show that *proof-based PRGs* where the input (i.e., the seed) is *entirely* chosen by the evaluator cannot be constructed in a black-box manner from an (ordinary) OWF chosen from the class of all OWFs. Since PRGs can be constructed in a fully-black-box manner from OWFs [GL89, ILL89, HILL99], this separates proof-based PRGs from ordinary PRGs.

More specifically, black-box construction of a proof-based PRG from (ordinary) OWFs consists of a *deterministic* and efficient oracle algorithm $G^{(\cdot)}$, along with an efficient protocol, $\Pi_G^{(\cdot)} = \langle P^{(\cdot)}, V^{(\cdot)} \rangle$, of two interactive oracle machines.[1] For every OWF f, algorithm G^f should be a PRG, and protocol $\Pi^f = \langle P^f, V^f \rangle$ should be a ZKP system for the relation $\mathcal{R}_G^f = \{(y, x) \mid \text{s.t. } y = G^f(x)\}$. Then, we show that a *fully-black-box* reduction [IR89, RTV04] from proof-based PRGs to ordinary OWFs does not exist if the prover chooses the entire seed.

The range-membership relation $\mathcal{R}^f = \{(y, x) \mid y = f(x)\}$ ruled out in [Ros12] is a special case of the aforementioned relation $\mathcal{R}_G^f = \{(y, x) \mid \text{s.t. } y = G^f(x)\}$. In our terminology, Rosulek rules out a special type of proof-based OWF $(F^{(\cdot)}, \Pi_F^{(\cdot)})$ where F is just a "delegate" for the oracle OWF; i.e., it returns the oracle's response when queried on the given input. This is captured in [Ros12] by formally defining the notion of *functionally-black-box* (FBB) protocols. In contrast, the relation we consider can make polynomially many queries to the oracle on arbitrary inputs and compute over the responses to produce the output. We extend the notion of FBB protocols to formally capture these extensions.

Partly due to these differences and our overall goals, our negative result is incomparable to Rosulek's. While Rosulek rules out black-box proofs for the range membership for OWFs assuming injective OWFs, ours is only a black-box separation, albeit without any additional assumptions. A black-box separation is the best one can hope for in our setting since *non-black-box* constructions of proof-based OWFs that use the code of the oracle trivially exist.

Positive Results. We next investigate whether mild restrictions on the inputs can help bypass the black-box separation result. One option is to consider modifications along the lines of the Goldreich-Levin (GL) hardcore predicate [GL89], where one considers a OWF F constructed from any given OWF f on inputs of the form (x, r). This makes it possible to show that predicate $\mathsf{hc}(x, r) := \oplus_i (x_i \cdot r_i)$ is hardcore for the

[1] Note that the protocol is allowed to depend on $G^{(\cdot)}$ but not on the oracle which may be arbitrarily chosen later. The same holds for the relation $\mathcal{R}_G^f$ introduced next.

modified function $F(x,r) := r \| f(x)$ even though a hardcore predicate for arbitrary OWFs is still unknown. These changes to the function and the input do not seem to affect the applicability of their result significantly.

We adopt a similar approach to construct proof-based primitives. Continuing with OWFs as an example, we seek to construct a proof-based OWF $F^{(\cdot)}$ which can be instantiated with only black-box access to any OWF f and takes inputs of the form (x,r). As in the GL setting, x will act as the "main input" chosen by the evaluator/prover, and r will be publicly accessible from the output of $F^f(x,r)$. However, in a crucial difference, r will be *chosen by the verifier* during the execution of ZKP Π_F^f. There are no other restrictions on any of the objects. Some remarks are in order.

1. In light of our black-box separation result, it is essential to let the verifier choose r since no other restrictions are present. This means that the computation of $y = F^f(x,r)$ must be performed during the proof. We formalize this by modeling Π_F^f as *a secure two-party computation* protocol for evaluating the functionality that on inputs x and r from relevant parties, returns y. The ZK property is captured by requiring simulation-based security against malicious receivers; for soundness, we only require that the honest verifier, with high probability, does not output a y^* that is not in the range. This is effectively a black-box ZKP for the relation $\mathcal{R}_F^f(r) = \{(y,x) \mid \text{s.t. } y = F^f(x,r)\}$.[2]
2. The verifier must choose r from an unpredictable distribution such as the uniform distribution over sufficiently long strings, since otherwise, the soundness would be impossible as a cheating prover can simply guess r, bringing us back to the setting of the separation result.
3. Since the verifier may maliciously choose r to violate the one-way property of F^f, we require that for *every string* r, the function defined by $F^f(\cdot, r)$ is one-way as long as f is one-way.

We follow the same approach for formally defining proof-based versions of PRGs and CRHFs. Having settled on a satisfactory definition, we present black-box constructions of the proof-based versions of OWFs (for range membership), as well as PRGs and CRHFs, directly from their ordinary counterparts.

Theorem 1 (Informal). *There is a fully-black-box construction of* proof-based primitive *as described above for the range-membership relation with two-party inputs of the form* (x,r)*, assuming that* primitive *exists, where* primitive $\in \{\text{OWF}, \text{PRG}, \text{CRHF}\}$.

At first glance, one may wonder whether black-box commit-and-prove protocols already yield proof-based OWFs. That is, the commit stage of such protocols can be viewed as a one-way function over the input (x,r) where x is the value to be committed and r is the randomness, the output y is the transcript of the commit-phase, and the proof-phase plays the role of associated ZKP. This approach does not work since the commit-and-prove protocols merely *bind* the prover to a well-defined value x. They do not guarantee that w.h.p. every accepting transcript has a valid "preimage" (x,r) that maps to it. In contrast, the soundness of range-membership proofs of proof-based OWFs requires that w.h.p. a preimage must exist for the output accepted by the honest verifier. At a technical level, the black-box commit-and-prove protocols are based on cut-and-choose techniques that can only guarantee that the accepted value is *close* to an honestly generated value, which is insufficient to guarantee a preimage.

Supporting Predicates. We show that it is possible to construct a slightly more general proof-system than merely range membership for each of our proof-based primitives. Continuing with the OWF example, we can construct a black-box proof-based OWF F^f such that for any predicate ϕ, the verifier learns a value y with the guarantee that there exists an input (x,r) such that: (1) $y = F^f(x,r)$ where r is chosen uniformly by the honest receiver, (2) $x = \alpha \| x'$, and (3) $\phi(\alpha) = 1$. That is, we can support any predicate (in fact, computation of any function) over a *prefix* of the preimage of the output. The ZKP system here depends on the code of ϕ but not that of f as before.

This extension is motivated by similar results for commit-and-prove, which are quite useful in constructing larger black-box protocols [GLOV12, KOS18]. We achieve this by presenting a new construction that

[2] For now, we only focus on range-membership proofs. The definitional approach is consistent with the commit-and-prove literature, although there are important differences since we are dealing with deterministic primitives.

combines our ideas for range-membership with the "MPC-in-the-head" technique [IKOS07]. More details are provided in Sec. 8.

2 Technical Overview

2.1 Black-Box Separation

We first present a very brief overview of our black-box separation. A detailed overview is given in Sec. 4.2 after setting up necessary notation and definitions.

Let us first recall how Rosulek [Ros12] rules out FBB constructions of honest-verifier witness-hiding (HVWH) protocols for the range-membership of OWFs, assuming injective OWFs exist.

The proof starts by assuming that such protocols exist. In particular, when instantiated with an injective OWF f, the protocol (P^f, V^f) is HVWH for $\mathcal{R}^f = \{(y, x) \mid \text{s.t. } y = f(x)\}$. Since f is injective, for a pair $(x^*, y^* = f(x^*))$ remapping $f(x^*)$ to a value different from y^* will give us a new OWF f' whose range does not contain y^* anymore. Moreover, the verifier accepts in $\langle P^f(x^*, y^*), V^{f'}(y^*)\rangle$ with roughly the same probability as in $\langle P^f(x^*, y^*), V^f(y^*)\rangle$. This is because the only opportunity to distinguish these two executions is when the verifier queries its oracle on x^*; but this happens with negligible probability because of the HVWH property of the protocol. However, this contradicts the soundness: V's oracle now becomes f', and $\nexists x$ s.t. $(y^*, x) \in \mathcal{R}^{f'}$.

It is unclear how to reuse the above technique to rule out PB-OWFs. As mentioned earlier, there are no restrictions on how the $F^{(\cdot)}$ part behaves. In particular, F^f is not guaranteed to be injective even if f is injective. Thus, "carving out" a value from the range of f may not affect the range of F^f.

To derive the desired contradiction, we take a fundamentally different approach to construct f'. We first define a set Q^{Easy}, which consists of only the queries made by the verifier with "high" probability during the (honest) execution $\langle P^f(x^*, y^*), V^f(y^*)\rangle$. We then define f' by maintaining the same behavior as f on Q^{Easy}, *and* re-sampling all the remaining points uniformly at random. Note that the receiver will still accept with "high" probability even if we change its oracle to f', because f' and f only differ at the points that are queried with "low" probability (i.e., the points outside Q^{Easy}). Now, the only thing left is to show that y^* is not in the range of $F^{f'}$. Unfortunately, due to the generality of $F^{(\cdot)}$, we do not know how to do that.

However, if we switch our focus to a PB-PRG G^f (instead of PB-OWF F^f), we can prove the following claim which helps separate PB-PRGs from OWFs:

Claim 1 (Informal Statement of Claim 4). *Let f, and f' be as defined above. If we start with a y^* in the range of G^f, y^* is still in the range of $G^{f'}$ with probability $\leq 0.5 + \mathsf{negl}(\lambda)$, where $G^{(\cdot)}$ is a PRG when instantiated with any OWF f as its oracle.*

Let us show the intuition behind Claim 1. Assume the lemma is false. In the pseudo-randomness game for G^f, we show how to identify the case $y^* \in \mathsf{Range}(G^f)$ *correctly*, with probability noticeably greater than 0.5, thus contradicting pseudo-randomness. To do that, the adversary simply estimates the probability that $y^* \in \mathsf{Range}(G^{f'})$. We will show that, if this probability is noticeably greater than 0.5, then $\Pr\left[y^* \in \mathsf{Range}(G^f)\right]$ is also noticeably greater than 0.5.

To perform the above estimation successfully, the adversary must know Q^{Easy}, but it does not. However, the adversary can run the HVZK simulator many times to get an estimate $\widetilde{Q}^{\mathsf{Easy}}$ for the real Q^{Easy}. We will show that $\widetilde{Q}^{\mathsf{Easy}}$ suffices for our proof. There is a caveat that the adversary needs to perform exponential work to estimate the probability that $y^* \in \mathsf{Range}(G^{f'})$, even if it knows the set $\widetilde{Q}^{\mathsf{Easy}}$. Fortunately, it only makes *polynomially many* oracle queries (when executing the HVZK simulator), which suffices for establishing a *fully*-black-box separation.

2.2 Proof-Based One-Way Functions (and PRGs)

Let us start by considering the following basic construction for PB-OWF (F^f, Π_F^f) over inputs of the form (x, r). The construction is based on the "cut-and-choose" technique, where the sender queries the oracle f on

"blocks" of x, and the receiver checks a size-t random subset (defined by r) of the responses. This method is not sound, since it can only guarantee that the sender's response is correct on most but not all blocks. We will handle this issue by introducing a new idea.

Basic Construction. PB-OWF (F^f, Π_F^f) handles inputs of the form (x, r). F^f computes as follows:

1. Parse x as $(x_1, \ldots, x_n)$.
2. Interpret r as a size-t ($t < n$) subset of $[n]$, denoted by $\{b_1, \ldots, b_t\}$.
3. Output $\mathsf{y} = (y_1, \ldots, y_n) \| (x_{b_1}, \ldots, x_{b_t}) \| \mathsf{r}$, where $y_i = f(x_i)$ for all $i \in [n]$.

On input x to S^f (the sender) and r to R^f (the receiver)[3], the execution $\langle S^f(\mathsf{x}), R^f(\mathsf{r}) \rangle$ is as follows:

1. S^f parses x as $(x_1, \ldots, x_n)$, and computes $(y_1, \ldots, y_n)$ via its oracle access to f (i.e., $y_i = f(x_i)$). It sends $(y_1, \ldots, y_n)$ to the receiver.
2. R^f sends its input r to S^f. Same as in F^f, the r specifies a size-t subset $\{b_1, \ldots, b_t\}$ of $[n]$. Recall that the honest receiver's input r is random. In this case, $\{b_1, \ldots, b_t\}$ is a *random* subset of $[n]$.
3. S^f sends $(x_{b_1}, \ldots, x_{b_t})$, i.e., the x_i's whose indices are specified by r.
4. R^f checks (via its oracle access to f) if $y_{b_i} = f(x_{b_i})$ for all $i \in [t]$. If all the checks pass, R^f output $\mathsf{y} = (y_1, \ldots, y_n) \| (x_{b_1}, \ldots, x_{b_t}) \| \mathsf{r}$.

Completeness is straightforward; furthermore, $F^f(\cdot, \mathsf{r})$ is trivially one-way for every r since $t < n$ and f is a OWF. Let us first consider the *honest-verifier zero-knowledge* (HVZK) property of protocol Π_F^f.

Recall that the ZK property is defined via the ideal/real paradigm for secure computation, and it requires simulation-security against malicious receivers. Thus, to prove the HVZK property, we need to show an ideal-world simulator Sim for the honest receiver. This is easy since the honest receiver will always use the given input r, which is uniformly distributed. More specifically, Sim works by sampling a uniform r by itself, sending the r to the ideal functionality, and receiving in turn the output $\mathsf{y} = (y_1, \ldots, y_n) \| (x_{b_1}, \ldots, x_{b_t}) \| \mathsf{r}$. With this y, Sim can easily generate a simulated transcript that is identically distributed to the real one.

ZK Against Malicious Receivers. The above simulation strategy does not work for malicious receivers, because they may not use the given input r. Therefore, the simulator needs to somehow extract the candidate input r^* from the malicious receiver. However, the receiver will not give out its r^* until the sender/simulator sends the $\{y_i\}_{i \in [n]}$ values.

We point out that this issue cannot be fixed using standard methods such as requiring the receiver to commit to r and to open it later. This is because later, we will introduce a pre-image editing condition and require that the sender's computation of F^f be consistent with this editing.

We therefore use a different idea. We modify the protocol to use a black-box commit-and-prove scheme $\Pi_{\mathsf{ZKCnP}} = (\mathsf{BBCom}, \mathsf{BBProve})$ with ZK property (see Def. 8). This scheme has a pair of simulators $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ that can be used to simulate the receiver's view in the commit stage and the prove stage respectively. Our new Π_F^f is the same as before except for the following changes:

- In Step 1, instead of sending y_i's as before, the sender commits to them using BBCom. Formally, the sender sets $\nu = (y_1, \ldots, y_n)$ and executes $\mathsf{BBCom}(\nu)$ with the receiver.
- In Step 3, the sender sends both $\{x_{b_i}\}_{i \in [n]}$ *and* the value ν. It then proves using $\mathsf{BBProve}$ that this ν is indeed the value committed in BBCom.

As before, the receiver needs $(y_1, \ldots, y_n)$ to execute Step 4. Now, these values are contained in ν, and $\mathsf{BBProve}$ guarantees that the sender cannot change ν.

With these modifications, we can prove the ZK property for malicious receivers as follows. The simulator starts by running Sim_1 (the commit-phase simulator) with the malicious receiver R^{*f}. In this way,

[3] Henceforth, we will call the two parties "sender" and "receiver" instead of "prover" and "verifier". This is because our ZKP is captured by considering a secure computation style definition for two parties.

the simulator can go through Step 1 smoothly, without knowing the actual $\{y_i\}_{i\in[n]}$ values. Then, it will receive the r^* from R^{*f}. The simulator sends r^* to the ideal functionality and receives back $\mathsf{y} = (y_1,\ldots,y_n)\|(x_{b_1},\ldots,x_{b_t})\|\mathsf{r}^*$. It sends $\nu = (y_1,\ldots,y_n)$ and $(x_{b_1},\ldots,x_{b_t})$ to the receiver. Then, instead of executing $\mathsf{BBProve}$, the simulator invokes Sim_2 to help itself go through the $\mathsf{BBProve}$ stage. It is easy to see that the ν and $\{x_{b_i}\}_{i\in[t]}$ sent by the simulator meet the consistency requirement in Step 4. Relying on the ZK property of Π_{ZKCnP}, one can formally prove that the simulation is done properly.

Soundness and Preimage Editing. As mentioned earlier, the "cut-and-choose" structure is not sufficient to guarantee the existence of a preimage. To see that, consider a malicious sender who picks an $i^* \in [n]$ at random, sets y_{i^*} to some value not in the range of f, or behaves honestly otherwise. This malicious sender can still make the honest receiver accept with non-negligible probability, even if t is as large as $n-1$ (the upper bound for t to achieve any non-trivial ZK property). This is addressed by modifying the construction of F^f.

We start by noting that the "cut-and-choose" trick ensures that most of the y_i's are "good" (i.e., having preimages under f). For example, if t is a constant fraction of n, then the protocol ensures (except with negligible probability) that at most k of the y_i's are "bad", where k is another constant fraction of n. Therefore, our idea is to extend the range of F^f to include all the images y that have $\leq k$ bad y_i's. More specifically, our new F^f works as follows. On input (x,r), it still interprets r as $\{b_1,\ldots,b_t\}$. But it will parse x as

$$\mathsf{x} = (x_1,\ldots,x_n)\|\underbrace{(p_1,y'_{p_1}),\ldots,(p_k,y'_{p_k})}_{\beta},$$

where the $\{p_1,\ldots,p_k\}$ form a size-k subset of $[n]$. The evaluation of $F^f(\mathsf{x},\mathsf{r})$ consists of two cases:

- **Non-Editing Case:** if $\{b_1,\ldots,b_t\}\cap\{p_1,\ldots,p_k\} \neq \emptyset$, then it computes y as before, ignoring the β part. That is, it outputs $\mathsf{y} = (s_1,\ldots,s_n)\|(x_{b_1},\ldots,x_{b_t})\|\mathsf{r}$, where $s_i = y_i$ for all $i \in [n]$.
- **Editing Case:** if $\{b_1,\ldots,b_t\}\cap\{p_1,\ldots,p_k\} = \emptyset$, then at positions specified by p_i's, it replace y_{p_i} with y'_{p_i}. Namely, it outputs $\mathsf{y} = (s_1,\ldots,s_n)\|(x_{b_1},\ldots,x_{b_t})\|\mathsf{r}$, where $s_i := \begin{cases} y'_i & i \in \{p_1,\ldots,p_k\} \\ y_i & i \in [n]\setminus\{p_1,\ldots,p_k\} \end{cases}$.

Let us explain how this editing technique resolves the soundness issue. Consider a y^* learned by the honest receiver with input r. As mentioned before, there are at most k y_i values (among those contained in y^*) that do not have preimages under f. These values can be expressed as $\{y^*_{p_1},\ldots,y^*_{p_k}\}$, i.e., their indices are $\{p_1,\ldots,p_k\}$. Moreover, this set of bad indices does not overlap with the $\{b_1,\ldots,b_t\}$ specified by r; otherwise, the receiver would abort when performing the checks in Step 4. Therefore, by setting the β part to $(p_1,y^*_{p_1}),\ldots,(p_k,y^*_{p_k})$, we will obtain a valid preimage for y^* under our new $F^f(\cdot,\mathsf{r})$.

One may wonder whether a malicious sender can cheat by taking advantage of the **editing** case. However, since the honest receiver will use a random r, the set $\{b_1,\ldots,b_t\}$ will always overlap with $\{p_1,\ldots,p_k\}$ (except with negligible probability). That is, although we prove soundness by relying on the **editing** case, it almost never happens in a real execution. So, this will not give malicious senders any extra power.

We remark that the above preimage-editing idea is compatible with our technique for achieving (full) ZK. Now, the sender will append $(y_1,\ldots,y_n)$ and β to the committed value ν. Upon receiving R^f's challenge r, the sender computes $\mathsf{s} = (s_1,\ldots,s_n)$ according to the above definition of F^f. It sends both s and $\{x_{b_1},\ldots,x_{b_t}\}$ to the receivers. Then, it runs $\mathsf{BBProve}$ to prove that it does the editing (or non-editing) honestly. Note that this statement can be expressed as a predicate on the values s, r, β, and $\{y_i\}_{i\in[n]}$, where the last two are committed in $\mathsf{BBCom}(\nu)$. Since it does not involve the code of f, the protocol remains black-box in f. We provide more details in Sec. 5.2.

Proof-Based PRGs. Following the above paradigm, we also obtain a proof-based PRG by simply replacing the oracle OWF f with a PRG in the above PB-OWF construction. We provide a formal treatment in Appx. A.

2.3 Proof-Based Collision-Resistant Hash Functions

Recall that a PB-CRHF consists of a function H^h and a protocol Π_H^h such that for any CRHF h:

- For all r, $H^h(\cdot, \mathsf{r})$ is a CRHF; **and**
- $\Pi_H^h = (S^h, R^h)$ is protocol satisfying similar completeness, soundness, and ZK properties as for our PB-OWFs, but w.r.t. H^h.

Let us first try to reuse the idea from our PB-OWFs. On input (x, r), the H^h first parses x as $(x_1, \ldots, x_n)\|\beta$, where the β has the same structure as before, for the purpose of preimage editing. It then generates $\{y_i\}_{i\in[n]}$ where $y_i = h(x_i)$, and outputs $\mathsf{y} = \mathsf{s}\|(x_{b_1}, \ldots, x_{b_t})\|\mathsf{r}$, where the value $\mathsf{s} = (s_1, \ldots, s_n)$ is computed by editing $\{y_i\}$ (in the same way as for our PB-OWFs).

Since h is also a OWF, the H^h is surely one-way. However, it is not collision-resistant. To see that, recall that in the **non-editing** case, the β part is not used when computing $H^h(\mathsf{x}, \mathsf{r})$. This implies the following collision-finding attack. For a fix r, the adversary first computes $\mathsf{y}^* = H^h(\mathsf{x}^*, \mathsf{r})$ with an x^* whose β part does *not* trigger the **editing** condition. Then, it can easily find many preimages for y^* by using different β's as long as they do not trigger the **editing** condition. Therefore, we need to come up with a new editing method that does not compromise collision resistance.

To do that, we modify H^h as follows. We sample a public string z and hard-wire it in H^h. In this way, H_z^h can be viewed as a member of the public-coin collision-resistant hash *family* indexed by z instead of a single CRHF. Then, we can think of x as containing additionally two strings τ and μ. When evaluating $H_z^h(\mathsf{x}, \mathsf{r})$, we will perform the editing if $\{b_1, \ldots, b_t\} \cap \{p_1, \ldots, p_k\} = \emptyset$ *and* $\alpha \neq z$ *and* $h(\tau) = h(z)$. Moreover, we include the value $\mathsf{t} = h(\beta\|\tau\|\mu)$ in the output y. Intuitively, this hash of β in y prevents the adversary from constructing collisions using a different β.

We now explain how to perform editing in this setting. First, we will include in x an additional value τ such that $\tau \neq z$ and $h(\tau) = h(z)$. This allows us to trigger the **editing** condition. With z sampled randomly, it is not hard to see that such a τ exists with overwhelming probability[4]. We can then set β as before to ensure that the $(x_1, \ldots, x_n)$ part is "edited" properly. However, note that the y^* here contains additionally a t^* value. To handle this, we modify the construction of H_z^f slightly—We require that, when the **editing** condition is triggered, H_z^f sets $\mathsf{t} = \mu$ in its output y. With this change, when performing editing, we can simply let μ equal the t^*. It is not hard to verify that this editing technique will lead to a valid preimage for y^*.

Finally, we remark that our real construction uses a Merkle tree for the prefix $(x_1, \ldots, x_n)$ of x. We only put the Merkle root in y instead of the element-wise hash values described above. The soundness can be proved following essentially the same idea as above, except that we now "edit" the Merkle tree, which is done by extending the editing ideas to the tree setting. This allows us to compress a prefix of any length to a *fixed-length* string, such as 256 bits if using SHA256 for h. We refer the reader to Sec. 7 for a formal treatment of PB-CRHF.

2.4 Supporting Predicates

We discuss how to extend our constructions using "MPC-in-the-head" to additionally guarantee not only that the output learned by the receiver is in the range of the deterministic primitives, but also that the set of preimages contains one whose prefix satisfies some predicate ϕ.

Let us take a fresh look at the PB-OWF construction. It first parses the input as $\mathsf{x} = \alpha\|\beta$. The β is for preimage editing; and the $\alpha = (x_1, \ldots, x_n)$ can be regarded as a form of **Encoding** the prefix of x, i.e. $\mathsf{Enc}(\alpha) = (x_1, \ldots, x_n)$. Then, it computes $y_i = f(x_i)$ for all $i \in [n]$. Since this is mainly to introduce hardness (or one-wayness) to the final output, we can refer to this step as **Hardness Inducing**.

[4] This holds if the size of range of h is exponentially larger than its image space. It is also worth noting that τ does not need to be efficiently computable, because our soundness proof (or the editing technique) is only an existential argument.

To support the proof of a predicate ϕ, we update the construction with new **Encoding** and **Hardness Inducing** methods. We first secret-share α to $([\alpha]_1, \ldots, [\alpha]_n)$ using a verifiable secret sharing (VSS) scheme. This can be viewed as a new encoding method: $\mathsf{Enc}(\alpha) = \mathsf{VSS}(\alpha) = ([\alpha]_1, \ldots, [\alpha]_n)$.

Next, we commit to these shares using Naor's commitment [Nao90], which can be built from the oracle OWF f in a black-box manner. This can be thought of as a new **Hardness Inducing** method. Now, the output of F^f is of the following form:

$$F^f(\mathsf{x}, \mathsf{r}) = (\mathsf{Com}([\alpha]_1), \ldots, \mathsf{Com}([\alpha]_n)) \| ([\alpha]_{b_1}, \ldots, [\alpha]_{b_t}) \| \mathsf{r}.$$

In the protocol Π^f_F, we additionally ask the sender to compute the value $\phi(\alpha)$ using the MPC-in-the-head technique. That is, the sender imagines n virtual parties $\{P_i\}_{i\in[n]}$, where P_i has $[\alpha]_i$ as its input. These n parties then execute a MPC protocol w.r.t. to the ideal functionality, which recovers α from the VSS shares, and outputs $\phi(\alpha)$ to each party. Let v_i denote the view of party i from the execution. The sender first commits to these views, and then opens some of them (picked by the receiver) for the receiver to check that the MPC for $\phi(\alpha)$ was performed honestly. In this way, the receiver not only learns $\phi(\alpha)$, but also believes that the sender did not cheat.

Finally, we make a few remarks:

- To achieve soundness, we also need to apply the preimage editing idea to the above construction.
- Both the VSS and Com require randomness, which can come from x. That is, we require that the x is long enough such that it also contains an η part (in addition to α and β). This η will provide the randomness for VSS and Com.
- The above approach applies directly to the PB-PRG and PB-CRHF constructions to make them support predicates on the α part of the preimage.

3 Preliminaries

Familiarity with basic cryptographic concepts such as ensembles, indistinguishability, and interactive Turing machines, etc. are assumed; we refer to [Gol01, Gol04] for formal treatments of these.

Notations. We use "\" to denote set difference. That is, for any two sets A and B, $A \backslash B \coloneqq \{x : (x \in A) \wedge (x \notin B)\}$. The security parameter is denoted by λ. Symbols $\overset{\mathrm{c}}{\approx}$, $\overset{\mathrm{s}}{\approx}$ and $\overset{\mathrm{i.d.}}{\equiv}$ are used to denote computational, statistical, and perfect indistinguishability respectively; and $\mathsf{negl}(\lambda)$ denotes negligible functions of λ. For a distribution D, $x \leftarrow D$ means that x is sampled according to D. Unless emphasized otherwise, we assume uniform distribution by default. We use $y \in D$ to mean that y is in the support of D. For a set S we overload the notation by using $x \leftarrow S$ to indicate that x is chosen uniformly at random from S. PPT denotes probabilistic polynomial time.

Let p be a predicate and $D_1, D_2, \ldots$ probability distributions, then the notation $\Pr\left[x_1 \leftarrow D_1; x_2 \leftarrow D_2; \ldots : p(x_1, x_2, \ldots)\right]$ denotes the probability that $p(x_1, x_2, \ldots)$ holds after the ordered execution of the probabilistic assignments $x_1 \leftarrow D_1; x_2 \leftarrow D_2; \ldots$. The notation $\{x_1 \leftarrow D_1; x_2 \leftarrow D_2; \ldots : p(x_1, x_2, \ldots)\}$ denotes the new probability distribution over $\{(x_1, x_2, \ldots)\}$.

3.1 Standard Primitives

Commitment Schemes. We will use Naor's two-round commitment scheme [Nao90], which is computationally-hiding and statistically-binding. It can be constructed from any OWFs in a black-box manner. The original scheme is for committing a single bit. But it can be extended to multi-bit messages by applying it bit-wise. We recall how this works. To commit a single bit b, R sends a string $\rho \in \{0,1\}^{3\lambda}$ to S. Then S picks a random seed $s \leftarrow \{0,1\}^\lambda$ and applies a pseudo-random generator $\mathsf{PRG}(\cdot)$ (which can be constructed from any OWFs in a black-box way). If $b = 0$, S sends $\mathsf{cm} = \mathsf{PRG}(s)$; if $b = 1$, S sends $\mathsf{cm} = \mathsf{PRG}(s) \oplus \rho$.

Collision-Resistant Hash Families. We use the definition from [HR04].

Definition 1 (Collision-Resistant Hash Families). *A private-coin collision-resistant hash family (CRHF) is a collection of functions $\mathcal{H} = \{h_i\}_{i \in I}$ for some index set I, where $h_i : \{0,1\}^{\ell(|i|)} \to \{0,1\}^{\ell'(|i|)}$ and $\ell(|i|) > \ell'(|i|)$. It satisfies the following requirements:*

- **Key Generation.** *There exists a PPT key generating algorithm* KGen*, so that* $\mathsf{KGen}(1^\lambda) \in \{0,1\}^{m(\lambda)} \cap I$*, where $m(\lambda)$ is a polynomial on λ representing the length of the key.*
- **Efficient Evaluation.** *There exists a (deterministic) polynomial time algorithm* Eval *such that $\forall i \in I$ and $\forall x \in \{0,1\}^{\ell(|i|)}$,* $\mathsf{Eval}(i, x) = h_i(x)$.
- **Non-Uniform Collision Resistance.** *For any non-uniform PPT machine $\mathcal{A}$, the following holds:*

$$\Pr\left[i \leftarrow \mathsf{KGen}(1^\lambda), (x, x') \leftarrow \mathcal{A}(i) : x \neq x' \wedge h_i(x) = h_i(x')\right] \leq \mathsf{negl}(\lambda).$$

Remark 1 (Public-Coin CRHFs). The CRHFs defined above is private-coin. In the above definition, if the index set I is $\{0,1\}^*$ and $\mathsf{KGen}(1^\lambda)$ outputs a uniformly distributed string from $\{0,1\}^{m(\lambda)}$, then we say that it is a *public-coin* CRHF, i.e., the family remains collision-resistant even if the randomness used to generate the key is known to the adversary. Also, we emphasize that the collision-resistance property holds against *non-uniform* PPT adversaries.

3.2 Verifiable Secret Sharing Schemes

A verifiable secret sharing (VSS) [CGMA85] scheme is a two-stage secret sharing protocol for implementing the following functionality. In the first stage, denoted by $\mathsf{VSS}_{\mathsf{Share}}$, a special player referred to as dealer, shares a secret s among n players, in the presence of at most t corrupted players. In the second stage, denoted by $\mathsf{VSS}_{\mathsf{Recon}}$, players exchange their views of the **Share** stage, and reconstruct the values. The functionality ensures that when the dealer is honest, before the second stage begins, the t corrupted players have no information about the secret. Moreover, when the dealer is dishonest, at the end of the **Share** stage the honest players would have realized it through an accusation mechanism that disqualifies the dealer.

The formal definition is presented in Def. 2. [BGW88, CDD+99] implemented $(n+1, \lfloor n/3 \rfloor)$-perfectly secure VSS schemes, and $(n+1, \lfloor n/4 \rfloor)$-perfectly secure VSS schemes can be found in [GIKR01]. These constructions suffices for our purpose in Sec. 8 where we only need that $t = o(n)$.

Definition 2 (Verifiable Secret Sharing). *An $(n+1, t)$-perfectly secure VSS scheme Π_{VSS} consists of a pair of protocols $(\mathsf{VSS}_{\mathsf{Share}}, \mathsf{VSS}_{\mathsf{Recon}})$ that implement respectively the sharing and reconstruction phases as follows.*

- **Sharing Phase $\mathsf{VSS}_{\mathsf{Share}}$:** *Player P_{n+1} (referred to as dealer) runs on input a secret s and randomness r_{n+1}, while any other player P_i ($i \in [n]$) runs on input a randomness r_i. During this phase players can send (both private and broadcast) messages in multiple rounds.*
- **Reconstruction Phase $\mathsf{VSS}_{\mathsf{Recon}}$:** *Each shareholder sends its view v_i ($i \in [n]$) of the sharing phase to each other player, and on input the views of all players (that can include bad or empty views) each player outputs a reconstruction of the secret s.*

All computations performed by honest players are efficient. The computationally unbounded adversary can corrupt up to t players that can deviate from the above procedures. The following security properties hold.

- **Commitment:** *if the dealer is dishonest, then one of the following two cases happen: 1) during the sharing phase honest players disqualify the dealer, therefore they output a special value $\perp$ and will refuse to play the reconstruction phase; 2) during the sharing phase honest players do not disqualify the dealer, therefore such a phase determines a unique value s^* that belongs to the set of possible legal values that does not include $\perp$, which will be reconstructed by the honest players during the reconstruction phase.*
- **Secrecy:** *if the dealer is honest, then the adversary obtains no information about the shared secret before running the protocol* Recon.
- **Correctness:** *if the dealer is honest throughout the protocols, then each honest player will output the shared secret s at the end of protocol* Recon.

3.3 MPC-in-the-Head

We first recall *information-theoretically secure* MPCs and relevant notion that will be employed in the MPC-in-the-head paradigm shown later.

Information-Theoretical MPC. Secure multi-party computation protocols allow several parties to compute a function f on their joint private inputs even in the presence of corrupted parties that try to glean other parties' inputs. Formally, we consider n different parties, of whom t parties are corrupted. Further, let $\mathcal{A}$ denote a real world adversary for protocol Π that recieves auxillary input z, and S denote an ideal world adversary for Π. Denote by $\mathsf{REAL}_{\Pi,\mathcal{A}(z),I}(\bar{x})$ the random variable consisting of the output of $\mathcal{A}$ controlling a set of corrupted parties I, and the outputs of the honest parties with respect to an execution of Π where party P_i has input x_i for $i \in [n]$, and $\bar{x} = (x_1, \ldots x_n)$. Similarly, denote by $\mathsf{IDEAL}_{f,S(z),I}(\bar{x})$ the corresponding output of S and other honest parties after an ideal execution with a trusted party computing f on inputs $\bar{x}$. We refer the reader to [Gol04] for a detailed description of the ideal and real executions.

Definition 3 (Perfectly/Statistically-Secure MPC). *Let $f : (\{0,1\}^*)^n \longrightarrow (\{0,1\}^*)^n$ be an n-ary functionality, and let Π be a protocol. We say that Π (n,t)-*perfectly (resp., statistically) securely *computes f if for every static, malicious, and (possibly-inefficient) probabilistic adversary $\mathcal{A}$ in the real model, there exists a probabilistic adversary S of comparable complexity (i.e., with runtime polynomial in that of $\mathcal{A}$) in the ideal model, such that for every $I \subset [n]$ of cardinality at most t, every $\bar{x} = (x_1, \ldots, x_n) \in (\{0,1\}^*)^n$ (where $|x_1| = \cdots = |x_n|$), and every $z \in \{0,1\}^*$, it holds that:*

$$\{\mathsf{REAL}_{\Pi,\mathcal{A}(z),I}(\bar{x})\} \stackrel{i.d.}{\equiv} \{\mathsf{IDEAL}_{f,S(z),I}(\bar{x})\} \quad \big(\textit{resp.}, \{\mathsf{REAL}_{\Pi,\mathcal{A}(z),I}(\bar{x})\} \stackrel{s}{\approx} \{\mathsf{IDEAL}_{f,S(z),I}(\bar{x})\}\big).$$

Recall that the MPC protocol from [BGW88] achieves (n,t)-perfect security (against static and malicious adversaries) with t being a constant fraction of n.

Theorem 2 ([BGW88]). *Consider a synchronous network with pairwise private channels. Then, for every n-ary functionality f, there exists a protocol that (n,t)-perfectly securely computes f in the presence of a static malicious adversary for any $t < n/3$.*

Consistency, Privacy, and Robustness. We now define some notation related to MPC protocols. Their roles will become clear when we discuss the MPC-in-the-head technique later.

Definition 4 (View Consistency). *A view View_i of an honest player P_i during an MPC computation Π contains input and randomness used in the computation, and all messages received from and sent to the communication tapes. We have that a pair of views $(\mathsf{View}_i, \mathsf{View}_j)$ are consistent with each other if*

1. *Both corresponding players P_i and P_j individually computed each outgoing message honestly by using the random tapes, inputs and incoming messages specified in View_i and View_j respectively, and:*
2. *All output messages of P_i to P_j appearing in View_i are consistent with incoming messages of P_j received from P_i appearing in View_j (and vice versa).*

We further define the notions of correctness, privacy and robustness for multiparty protocols.

Definition 5 (Semi-Honest Computational Privacy). *Let $1 \leq t < n$, let Π be an MPC protocol, and let $\mathcal{A}$ be any* static, PPT, and semi-honest *adversary. We say that Π realizes a function $f : (\{0,1\}^*)^n \longrightarrow (\{0,1\}^*)^n$ with* semi-honest (n,t)-computational privacy *if there is a PPT simulator Sim such that for any inputs $x, w_1, \ldots, w_n$, every subset $T \subset [n]$ ($|T| \leq t$) of players corrupted by $\mathcal{A}$, and every D with circuit size at most $\mathsf{poly}(\lambda)$, it holds that*

$$\big|\Pr[D(\mathsf{View}_T(x, w_1, \ldots, w_n)) = 1] - \Pr[D(\mathsf{Sim}(T, x, \{w_i\}_{i\in T}, f_T(x, w_1, \ldots, w_n))) = 1]\big| \leq \mathsf{negl}(\lambda), \qquad (1)$$

where $\mathsf{View}_T(x, w_1, \ldots, w_n)$ is the joint view of all players.

Definition 6 (Statistical/Perfect Correctness). *Let Π be an MPC protocol. We say that Π realizes a deterministic n-party functionality $f(x, w_1, \ldots, w_n)$ with* perfect (resp., statistical) *correctness if for all inputs $x, w_1, \ldots, w_n$, the probability that the output of some party is different from the output of some party is different from the actual output of f is 0 (resp., negligible in k), where the probability is over the independent choices of the random inputs $r_1, \ldots, r_n$ of these parties.*

Definition 7 (Perfect/Statistical Robustness). *Assume the same setting as the previous definition. We say that Π realizes f with (n,t)-*perfect (resp., statistical) robustness *if in addition to being perfectly (resp., statistical) correct in the presence of a semi-honest adversary as above, it enjoys the following* robustness *property against any computationally unbounded malicious adversary corrupting a set T of at most t parties, and for any inputs $(x, w_1, \ldots, w_n)$: if there is no $(w'_1, \ldots, w'_n)$ such that $f(x, w'_1, \ldots, w'_n) = 1$, then the probability that some uncorrupted player outputs 1 in an execution of Π in which the inputs of the honest parties are consistent with $(x, w_1, \ldots, w_n)$ is 0 (resp., negligible in λ).*

MPC-in-the-Head. MPC-in-the-head is a technique developed for constructing black-box ZK protocols from MPC protocols [IKOS07]. Very roughly, the MPC-in-the-head idea is the following. Let $\mathcal{F}_{\text{ZK}}$ be the zero-knowledge functionality for an NP language. $\mathcal{F}_{\text{ZK}}$ takes as public input x and one share from each party, and outputs 1 iff the secret reconstructed from the shares is a valid witness. To build a ZK protocol, the prover runs in his head an execution of MPC w.r.t. $\mathcal{F}_{\text{ZK}}$ among n imaginary parties, each one participating in the protocol with a share of the witness. Then, it commits to the view of each party separately. The verifier obtains t randomly chosen views, checks that such views are "consistent" (see Def. 4), and accepts if the output of every party is 1. The idea is that, by selecting the t views at random, V will catch inconsistent views if the prover cheats.

We emphasize that, in this paradigm, a malicious prover decides the randomness of each virtual party, including those not checked by the verifier (corresponding to honest parties in the MPC execution). Therefore, MPC protocols with standard computational security may not protect against such attacks. We need to ensure that the adversary cannot force a wrong output even if it additionally controls the honest parties' random tape. The $(n, \lfloor n/3 \rfloor)$-perfectly secure MPC protocol in Thm. 2 suffices for this purpose (see also Rmk. 2).

One can extend this technique further (as in [GLOV12]), to prove a general predicate ϕ about an arbitrary value α. Namely, one can consider the functionality $\mathcal{F}_\phi$ in which party i participates with input a VSS share $[\alpha]_i$. $\mathcal{F}_\phi$ collects all such shares, and outputs 1 iff $\phi(\mathsf{VSS}_{\mathsf{Recon}}([\alpha]_1, \ldots, [\alpha]_n)) = 1$.

Remark 2 (Exact Security Requirements on the Underlying MPC.). To be more accurate, any MPC protocol that achieves *semi-honest (n,t)-computational privacy (see Def. 5) and (n,t)-perfect robustness (see Def. 7)* will suffice for the MPC-in-the-head application.[5] These two requirements are satisfied by any (n,t)-perfectly secure MPC (and, in particular, the one from Thm. 2).

3.4 Black-Box Zero-Knowledge Commit-and-Prove

We need a zero-knowledge commit-and-prove protocol Π_{ZKCnP} with the following additional properties:

- it consists of two *separate* phases: a **Commit** stage BBCom and a **Prove** stage $\mathsf{BBProve}$;
- the **Commit** stage itself constitutes a statistically-binding commitment scheme;
- for a public predicate $\phi(\cdot)$, the **Prove** stage constitutes a zero-knowledge argument for the value $\phi(x)$, where x is the value committed in BBCom;
- Π_{ZKCnP} can be constructed assuming only black-box access to OWFs.

We present the formal definition in Def. 8. There exist constructions satisfying this definition while making only black-box use of OWFs (e.g., [IKOS07, GLOV12, CLP20]).

[5] It is also worth noting that the (n,t)-perfect robustness could be replaced with *adaptive (n,t)-statistical robustness*. See [IKOS07, Section 4.2] for more details.

Definition 8 (Zero-Knowledge Commit-and-Prove). *A black-box zero-knowledge commit-and-prove scheme consists of a pair of protocols* $\Pi_{\mathsf{ZKCnP}} = (\mathsf{BBCom}, \mathsf{BBProve})$ *executed between a pair of* PPT *machines* P *and* V*. These protocols are executed in the following phases:*

- **Commit Stage:** *P and V invoke* $\mathsf{BBCom}(x)$ *such that at the end of this protocol, P is statistically committed to the value x. We use τ to denote the transcript from* $\mathsf{BBCom}(1^\lambda, x)$ *execution. P stores private state* ST*.*
- **Prove Stage:** *P and V take the transcript τ and a predicate ϕ as common input. P takes* ST *as its private input. P proves to V using* $\mathsf{BBProve}(1^\lambda, \phi)$ *that there exists some value x such that τ is a valid commitment to x, and also $\phi(x) = 1$.*

We require that the following properties are satisfied:

- **Black-Box.** *Both phases only require black-box access to cryptographic primitives.*
- **Committing.** *The* **Commit** *stage* $\mathsf{BBCom}(x)$ *constitutes a statistically-binding commitment to the value x.*
- **Completeness.** *If P and V are honest and $\phi(x) = 1$, then V accepts the proof with probability 1.*
- **Soundness.** *For any PPT prover P^*, the following holds except with negligible probability: V accepts only if $\phi(x) = 1$, where x is the value to which the* **Commit** *stage is statistically bound.*
- **Zero-Knowledge.** *There exists a pair of* PPT *machines* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *satisfying the following requirement. Given oracle access to a machine V^*,* $\mathsf{Sim}_1^{V^*}(1^\lambda)$ *generates a transcript* $\widetilde{\tau}$ *and stores a private state* ST*; then for any predicate ϕ,* $\mathsf{Sim}_2^{V^*}(1^\lambda, \phi, \mathsf{ST})$ *generates a transcripts* View*. For every* PPT *verifier V^*, every $z \in \{0,1\}^*$, every PPT predicate ϕ and every x s.t. $\phi(x) = 1$, it holds that*

$$\left\{\left(\mathsf{Sim}_1^{V^*}(1^\lambda),\ \mathsf{Sim}_2^{V^*}(1^\lambda, \phi, \mathsf{ST}, z)\right)\right\}_{\lambda\in\mathbb{N}} \quad \overset{c}{\approx} \quad \{\mathsf{View}_{V^*}^{P(x)}(1^\lambda, \phi, z)\}_{\lambda\in\mathbb{N}},$$

where the ST *is the private state of* $\mathsf{Sim}_1^{V^*}(1^\lambda)$ *at termination, and* $\mathsf{View}_{V^*}^{P(x)}(1^\lambda, \phi, z)$ *denotes the view of $V^*(1^\lambda, \phi, z)$ in both* **Commit** *stage and* **Prove** *stage, resulted from its interaction with $P(1^\lambda, x, \phi)$, where both parties learn ϕ at the beginning of* **Prove** *stage. We will refer to* Sim_1 *as the* **Commit***-stage simulator, and* Sim_2 *the* **Prove***-stage simulator.*

3.5 The One-Oracle Separation Technique

We first recall in Def. 9 the notion of *fully-black-box* reductions. We say that P cannot be obtained from Q in a fully-black-box way if there is no fully-black-box reduction from Q to P.

Definition 9 (Fully-Black-Box Reductions [RTV04]). *There exists a fully-black-box reduction from a primitive Q to a primitive P, if there exist PPT oracle machines G and S such that:*

- **Correctness:** *For every (possibly-inefficient) f that implements P, G^f implements Q;*
- **Security:** *For every (possibly-inefficient) f that implements P and every (possibly-inefficient) machine $\mathcal{A}$, if $\mathcal{A}$ breaks G^f (w.r.t. Q-security), then $S^{\mathcal{A},f}$ breaks f (w.r.t. P-security).*

A useful paradigm to rule out fully-black-box constructions is to design an oracle $\mathcal{O}$, and show that, relative to $\mathcal{O}$, primitive P exists but Q does not. A critical step in this proof is to construct an oracle machine $\mathcal{A}^{\mathcal{O}}$ that breaks the security of Q. We emphasize that $\mathcal{A}$ is allowed to be *computationally unbounded* as long as it only makes polynomially-many queries to $\mathcal{O}$ (see e.g., [IR89, BM17]). Our fully-black-box separation results in Sec. 4 will follow this paradigm.

4 The Impossibility Results

4.1 Meta-Functionally Black-Box Constructions

Functionally Black-Box Protocols. To capture MPC protocols that "do not know" the code of the target function g, Rosulek [Ros12] proposed the following notion of functionally-black-box protocols.

Definition 10 (Functionally-Black-Box Protocols [Ros12]). *Let $\mathcal{C}$ be a class of functions, and let $\mathcal{F}^{(\cdot)}$ be an ideal functionality that is an (uninstantiated) oracle machine. Let $A^{(\cdot)}$ and $B^{(\cdot)}$ be PPT interactive oracle machines. Then, we say that $(A^{(\cdot)}, B^{(\cdot)})$ is a functionally-black-box (FBB) protocol for $\mathcal{F}^{\mathcal{C}}$ in a certain security model if, for all $g \in \mathcal{C}$, the protocol (A^g, B^g) is a secure protocol (in the model in question) for the ideal functionality $\mathcal{F}^g$.*

By instantiating $\mathcal{C}$ and $\mathcal{F}^{(\cdot)}$ properly, Def. 10 could capture black-box constructions of many useful cryptographic protocols. For example, let $\mathcal{C}_{\text{OWF}}$ be the collection of OWFs. For any $g \in \mathcal{C}_{\text{OWF}}$, let $\mathcal{F}_{\text{ZK}}^g$ be the functionality that takes input x from party A, queries its oracle g to obtain $y = g(x)$, and outputs y to party B. Such an $\mathcal{F}_{\text{ZK}}^g$ is essentially a zero-knowledge argument (of knowledge) functionality for statements of the form "$\exists x$ s.t. $g(x) = y$". However, Rosulek showed that if injective OWFs exist, then it is impossible to have FBB protocols that implement $\mathcal{F}_{\text{ZK}}^{\mathcal{C}_{\text{OWF}}}$ with semi-honest security (in the standard MPC setting), even in the presence of an arbitrary trusted setup. Given the broad application of ZK proofs, this result is quite discouraging.

Meta-FBB Functionalities. Observe that the above $\mathcal{F}_{\text{ZK}}^g$ functionality simply collects input x from A, queries its oracle g, and sends $g(x)$ to B. It only plays the role of a delegate for A and B to interact with the OWF g. Therefore, it is temping to investigate whether we can circumvent Rosulek's lower bound by allowing the "delegate" $\mathcal{F}_{\text{ZK}}$ to perform extra computations, such as preprocessing x, post-processing $g(x)$, or making multiple queries to the oracle g, etc.

More formally, we want a non-cryptographic and deterministic computation F (used to capture the aforementioned extra computations), such that $\mathcal{C}'_{\text{OWF}} = \{F^g \mid g \in \mathcal{C}_{\text{OWF}}\}$ is a collection of OWFs. And we hope that there exists a FBB protocol (A^g, B^g) implementing $\mathcal{F}_{\text{ZK}}^{F^g}$ for all $F^g \in \mathcal{C}'_{\text{OWF}}$ (we can also denote it as $\mathcal{F}_{\text{ZK}}^{\mathcal{C}'_{\text{OWF}}}$). Note that we require $(A^{(\cdot)}, B^{(\cdot)})$ to access g in a black-box way only; they can make use the code of F. Since $\mathcal{C}'_{\text{OWF}}$ is also a collection of one-way families, $\mathcal{F}_{\text{ZK}}^{\mathcal{C}'_{\text{OWF}}}$ can be used as a substitute for $\mathcal{F}_{\text{ZK}}^{\mathcal{C}_{\text{OWF}}}$, with the only overhead coming from the computations represented by $F^{(\cdot)}$. Because $F^{(\cdot)}$ is supposed to contain only simple non-cryptographic operations, the implementation of $\mathcal{F}_{\text{ZK}}^{\mathcal{C}'_{\text{OWF}}}$ should be as efficient as that of $\mathcal{F}_{\text{ZK}}^{\mathcal{C}_{\text{OWF}}}$. Therefore, if this approach is possible, it will alleviate the negative implications of Rosulek's lower bound.

We can also interpret $\mathcal{F}_{\text{ZK}}^{\mathcal{C}'_{\text{OWF}}}$ as a new FBB functionality $\mathcal{F}_{\text{ZK}}^{\mathcal{C}_{\text{OWF}}}[F]$, i.e., a new oracle machine $\mathcal{F}_{\text{ZK}}^{(\cdot)}[F]$ to be instantiated with oracle OWFs from the original collection $\mathcal{C}_{\text{OWF}}$. For any $g \in \mathcal{C}_{\text{OWF}}$, $\mathcal{F}_{\text{ZK}}^g[F]$ collects the input X from Party A, evaluates $F^g(X)$, and sends $y = F^g(X)$ to Party B.

With this interpretation, $\mathcal{F}_{\text{ZK}}^{\mathcal{C}'_{\text{OWF}}}$ is just an instantiation of Def. 10 with $\mathcal{F}^{(\cdot)} = \mathcal{F}_{\text{ZK}}^{(\cdot)}[F]$ and $\mathcal{C} = \mathcal{C}_{\text{OWF}}$. To distinguish with Rosulek's $\mathcal{F}_{\text{ZK}}^{(\cdot)}$ functionality. We call $\mathcal{F}_{\text{ZK}}^{(\cdot)}[F]$ the *Meta-FBB* ZK Functionality. Similarly, one can also extend other FBB functionalities in [Ros12] (e.g., 2-party secure function evaluation $\mathcal{F}_{\text{SFE}}^{(\cdot)}$, pseudo-random generator $\mathcal{F}_{\text{PRG}}^{(\cdot)}$, where sender A holds the seed and receiver B holds the key) to the corresponding Meta-FBB version.

4.2 The Main Theorem

In this part, we show that although we relax Rosulek's FBB notion to the Meta-FBB one, there still exists strong impossibility result. More specifically, we prove that, given only black-box access to OWFs, it is impossible to build a PRG that admits Meta-FBB honest-verifier zero-knowledge protocols.

Definition 11 (Fully-Black-Box PRGs from OWFs). *Let $\mathcal{C}$ be the collection of OWFs. A (deterministic) polynomial-time oracle machines $G^{(\cdot)}$ is a* fully-black-box *construction of PRG from OWF if there exists a PPT oracle machines $\mathcal{A}^{(\cdot,\cdot)}$ such that:*

– **Correctness:** $\forall f \in \mathcal{C}$, G^f *is a PRG;*

– **Security:** $\forall f \in \mathcal{C}$ *and every (possibly inefficient) machine* M*, if* M *breaks the pseudo-randomness of* G^f*, then* $\mathcal{A}^{M,f}$ *breaks the one-wayness of* f*.*

Theorem 3 (Main Theorem). *Let* $\mathcal{C} = \{f \mid f \text{ is a OWF}\}$*. There does not exist a (deterministic) oracle machine* $G^{(\cdot)}$ *such that*

1. $G^{(\cdot)}$ *is a fully-black-box construction of PRG from OWF;* **and**
2. *for all* $f \in \mathcal{C}$*, there exists a stand-alone, Meta-FBB, honest-verifier zero-knowledge argument system* $\Pi^f = \langle P^f, V^f \rangle$ *for the functionality* $\mathcal{F}_{\text{ZK}}^f[G]$*.*

Before showing the full proof in Sec. 4.3, let us provide the high-level idea.

Proof Overview. We start by assuming (for contradiction) that the $G^{(\cdot)}$ and $\Pi^{(\cdot)}$ specified in the theorem exist. We will construct a special oracle denoted as $\mathsf{O} \diamond Q^{\mathsf{Easy}}$ (explained later) such that:

1. The oracle $\mathsf{O} \diamond Q^{\mathsf{Easy}}$ is one-way. Thus, $G^{\mathsf{O} \diamond Q^{\mathsf{Easy}}}$ will be a PRG and $\Pi^{\mathsf{O} \diamond Q^{\mathsf{Easy}}}$ will be the HVZK system for the language $\mathcal{L} = \{Y : \exists X \ s.t. \ Y = G^{\mathsf{O} \diamond Q^{\mathsf{Easy}}}(X)\}$.
2. There exist a $\ddot{Y} \notin \mathcal{L}$ (the false statement) and a $\mathbb{P}^{\mathsf{O} \diamond Q^{\mathsf{Easy}}}$ (the cheating prover $\mathbb{P}$ with the oracle $\mathsf{O} \diamond Q^{\mathsf{Easy}}$) that is able to make $V^{\mathsf{O} \diamond Q^{\mathsf{Easy}}}(\ddot{Y})$ accept.

This will give us the desired contradiction as it breaks the soundness of the protocol $\Pi^{\mathsf{O} \diamond Q^{\mathsf{Easy}}}$.

Toward the above goal, we first sample two random oracles O, O', a random string X, and compute $Y = G^{\mathsf{O}}(X)$. Let $Q = \{(q_1, \mathsf{O}(q_1)), \ldots, (q_t, \mathsf{O}(q_t))\}$ denote the query-answer pairs exchanged between G and its oracle O during computation $Y = G^{\mathsf{O}}(X)$. We now define the oracle $\mathsf{O}' \diamond Q(q) \coloneqq \begin{cases} \mathsf{O}(q) & \text{if } (q, \mathsf{O}(q)) \in Q \\ \mathsf{O}'(q) & \text{otherwise} \end{cases}$.

It is not hard to verify that $Y = G^{\mathsf{O}' \diamond Q}(X)$. By completeness, V will accept with probability $1 - \delta_c$ (where δ_c is the completeness error) in the execution $\mathsf{Exec}_{X,Y}^{\mathsf{O}' \diamond Q} = \langle P^{\mathsf{O}' \diamond Q}(X, Y), V^{\mathsf{O}' \diamond Q}(Y) \rangle$.

Note that during $\mathsf{Exec}_{X,Y}^{\mathsf{O}' \diamond Q}$, the verifier may make queries to its oracle $\mathsf{O}' \diamond Q$. We define a set of "easy" queries:

$$Q^{\mathsf{Easy}} \coloneqq \{(q, \mathsf{O}(q)) \mid V \text{ queries } q \text{ with "high" probability during } \mathsf{Exec}_{X,Y}^{\mathsf{O}' \diamond Q}\}.$$

Let Q^{Hard} be the set difference $Q \setminus Q^{\mathsf{Easy}}$. It is not hard to see that $Y = G^{\mathsf{O}' \diamond (Q^{\mathsf{Easy}} \cup Q^{\mathsf{Hard}})}(X)$. By completeness, V will accepts with probability $1 - \delta_c$ in the execution $\mathsf{Exec}_{X,Y}^{\mathsf{O}' \diamond (Q^{\mathsf{Easy}} \cup Q^{\mathsf{Hard}})}$.

Now, consider the execution $\langle P^{\mathsf{O}' \diamond (Q^{\mathsf{Easy}} \cup Q^{\mathsf{Hard}})}(X, Y), V^{\mathsf{O}' \diamond Q^{\mathsf{Easy}}}(Y) \rangle$, which is identical to $\mathsf{Exec}_{X,Y}^{\mathsf{O}' \diamond (Q^{\mathsf{Easy}} \cup Q^{\mathsf{Hard}})}$, except that we remove the Q^{Hard} from the verifier's oracle. In this execution, the probability that V accepts will not differ too much from that in $\mathsf{Exec}_{X,Y}^{\mathsf{O}' \diamond (Q^{\mathsf{Easy}} \cup Q^{\mathsf{Hard}})}$, because the queries in Q^{Hard} are asked by V with only "low" probability.

We then prove that Y is in the range of $G^{\mathsf{O}' \diamond Q^{\mathsf{Easy}}}(\cdot)$ with probability at most 0.5 (up to negligible error). But the previous argument says that $V^{\mathsf{O}' \diamond Q^{\mathsf{Easy}}}(Y)$ accepts with probability close to 1. It then follows from an averaging argument that there exists "bad" $\ddot{\mathsf{O}}$, $\ddot{\mathsf{O}}'$ and $\ddot{X}$[6] such that $\ddot{Y} = G^{\ddot{\mathsf{O}}}(\ddot{X})$ is *not* in the range of $G^{\ddot{\mathsf{O}}' \diamond \ddot{Q}^{\mathsf{Easy}}}(\cdot)$, but $V^{\ddot{\mathsf{O}}' \diamond \ddot{Q}^{\mathsf{Easy}}}(\ddot{Y})$ can be convinced with probability close to 1, by the malicious prover $P^{\ddot{\mathsf{O}}' \diamond (\ddot{Q}^{\mathsf{Easy}} \cup \ddot{Q}^{\mathsf{Hard}})}(\ddot{X}, \ddot{Y})$ (which can be viewed as an oracle machine $\mathbb{P}^{\ddot{\mathsf{O}}' \diamond \ddot{Q}^{\mathsf{Easy}}}$ with non-uniform advice $\ddot{X}$, $\ddot{Y}$, and $\ddot{Q}^{\mathsf{Hard}}$). This breaks the soundness of $\Pi^{\ddot{\mathsf{O}}' \diamond \ddot{Q}^{\mathsf{Easy}}}$, thus completing the proof.

We remark that proving Y is in the range of $G^{\mathsf{O}' \diamond Q^{\mathsf{Easy}}}(\cdot)$ with probability ≤ 0.5 (up to negligible error) is the most involved part. And this is where the HVZK property of $\Pi^{(\cdot)}$ plays an essential role. Roughly, we will show that if this claim does not hold, then there exists an adversary $\mathcal{A}_{\text{PRG}}^{\mathsf{O}}$ that can break the pseudo-randomness of $G^{\mathsf{O}}(\cdot)$ by making *polynomially* many oracle queries. As we will explain later, this reduction

[6] Note that these values already determine the sets $\ddot{Q}$, $\ddot{Q}^{\mathsf{Easy}}$, and $\ddot{Q}^{\mathsf{Hard}}$ as defined above.

requires $\mathcal{A}^{\mathsf{O}}_{\mathrm{PRG}}$ to know the set Q^{Easy} w.r.t. the challenge string Y in the security game of PRG. But note that $\mathcal{A}^{\mathsf{O}}_{\mathrm{PRG}}$ does not know the preimage X (if Y is indeed in the range), which is necessary to figure out Q^{Easy}. This is where the HVZK simulator comes to our rescue. We will run the simulator $\mathsf{Sim}^{\mathsf{O}}_V(Y)$ repeatedly for (polynomially) many times to get an estimate $\widetilde{Q}^{\mathsf{Easy}}$ for the set Q^{Easy}. This $\widetilde{Q}^{\mathsf{Easy}}$ will be good enough to finish our proof. A more detailed overview of this strategy is provided at the beginning of Sec. 4.4.

4.3 Proof of Thm. 3

Assume for contradiction that there exists an oracle machine $G^{(\cdot)}$ and a protocol $\langle P^{(\cdot)}, V^{(\cdot)}\rangle$ such that given the access to any one-way function $\{f_n\}_{n\in\mathbb{N}}$:

1. $G^{f_n} : \{0,1\}^{\ell} \to \{0,1\}^{\ell+1}$ is a PRG (ℓ and n are polynomially related); **and**
2. $\Pi = \langle P^{f_n}, V^{f_n}\rangle$ is a semi-honest zero-knowledge argument system for the Meta-FBB functionality $\mathcal{F}^{f_n}_{\mathrm{ZK}}[G]$.

We first recall the following lemma, which says that the measure-one of randomly-sampled oracles is one-way.

Lemma 1 (One-Wayness of Random Oracles [IR89, Yer11]). *Let $\mathcal{O} = \{\mathsf{O}_n\}_{n\in\mathbb{N}}$ be a collection of oracles where each O_n is chosen uniformly from the space of functions from $\{0,1\}^n$ to $\{0,1\}^n$. With probability 1 over the choice of $\mathcal{O}$, $\mathcal{O}$ is one-way against unbounded adversaries that make only polynomially many oracle queries to $\mathcal{O}$.*

Let both $\mathcal{O} = \{\mathsf{O}_n\}_{n\in\mathbb{N}}$ and $\mathcal{O}' = \{\mathsf{O}'_n\}_{n\in\mathbb{N}}$ be defined (independently) as in Lem. 1. It follows from Lem. 1 that, with probability 1, both $\mathcal{O}$ and $\mathcal{O}'$ are one-way.

In the following, we show two hybrids. From the second hybrid, we will construct a malicious prover breaking the soundness of $\Pi^{(\cdot)}$ (with the oracle being instantiated by a special one-way oracle defined later). This will give us the desired contradiction, and thus will finish the proof of Thm. 3.

Notation. We first define some notation. For an oracle H and a set of tuples $\mathsf{S} = \{(q_1, a_1), \ldots, (q_t, a_t)\}$, we define a new oracle $\mathsf{H} \diamond \mathsf{S}$ as follows: if q equals some q_i for which there exists a pair (q_i, a_i) in the set S, the oracle $\mathsf{H} \diamond \mathsf{S}$ returns a_i; otherwise, it returns $\mathsf{H}(q)$. Formally,

$$\mathsf{H} \diamond \mathsf{S}(q) = \begin{cases} \mathsf{H}(q) & \text{if } q \notin \{q_1, \ldots, q_t\} \\ a_i & \text{if } q = q_i \in \{q_1, \ldots, q_t\} \end{cases}.$$

Hybrid H_0. This hybrid samples $X_n \leftarrow \{0,1\}^{\ell(n)}$, and computes $Y_n = G^{\mathsf{O}_n}(X_n)$. W.l.o.g., we assume that G on input X_n makes $t(n)$ *distinct* queries to its oracle O_n, where $t(n)$ is a polynomial of n. Let $Q_n = \{(q_1, \mathsf{O}_n(q_1)), \ldots, (q_t, \mathsf{O}_n(q_t))\}$ be the query-answer pairs during the computation $Y_n = G^{\mathsf{O}_n}(X_n)$.

Let $\mathsf{Exec}^{\mathsf{O}'_n \diamond Q_n}_{X_n, Y_n} = \langle P^{\mathsf{O}'_n \diamond Q_n}(X_n, Y_n), V^{\mathsf{O}'_n \diamond Q_n}(Y_n)\rangle$ denote the execution where P proves to V that there exists an X_n such that $Y_n = G^{\mathsf{O}'_n \diamond Q_n}(X_n)$. Note that during this execution, the verifier may query its oracle $\mathsf{O}'_n \diamond Q_n$. For each $q_i \in \{0,1\}^n$, let p_i denote the probability that q_i is queried by V during $\mathsf{Exec}^{\mathsf{O}'_n \diamond Q_n}_{X_n, Y_n}$. Let Q^{Easy}_n defines the set of "easy" queries and their corresponding answers:

$$Q^{\mathsf{Easy}}_n := \left\{ (q_i, \mathsf{O}'_n \diamond Q_n(q_i)) \;\middle|\; p_i \geq \frac{1}{t(n) \cdot n} \text{ during } \mathsf{Exec}^{\mathsf{O}'_n \diamond Q_n}_{X_n, Y_n} \right\}. \tag{2}$$

Let Q^{Hard}_n be the set difference $Q_n \setminus Q^{\mathsf{Easy}}_n$. We remark that Q_n and $Q^{\mathsf{Easy}}_n \cup Q^{\mathsf{Hard}}_n$ may not be the same, but it must hold that $Q_n \subseteq Q^{\mathsf{Easy}}_n \cup Q^{\mathsf{Hard}}_n$.

Looking ahead, we will instantiate $G^{(\cdot)}$ and $\Pi^{(\cdot)}$ with the oracle $\mathsf{O}'_n \diamond (Q^{\mathsf{Easy}}_n \cup Q^{\mathsf{Hard}}_n)$. Note that $G^{(\cdot)}$ and $\Pi^{(\cdot)}$ will have the desired property only if they are instantiated with *one-way* functions. Therefore, we show in Claim 2 that the composed oracle $\mathsf{O}'_n \diamond (Q^{\mathsf{Easy}}_n \cup Q^{\mathsf{Hard}}_n)$ is one-way. It is worth noting that the one-wayness of this composed oracle is independent of the choice of $\{X_n\}$, though the definition of Q_n, Q^{Easy}_n and Q^{Hard}_n depends on X_n.

Claim 2. *The collection of oracles* $\left\{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})\right\}_{n\in\mathbb{N}}$ *defined above is one-way with probability 1, where the probability is taken over the sampling of* $\mathcal{O} = \{\mathsf{O}_n\}_n$ *and* $\mathcal{O}' = \{\mathsf{O}'_n\}_n$*, and is* independent *of the distribution of* $\{X_n\}_{n\in\mathbb{N}}$*.*

Proof. The query-answer pairs in Q_n^{Easy} and Q_n^{Hard} are of the form $\big(q, \mathsf{O}_n(q)\big)$ or $\big(q, \mathsf{O}'_n(q)\big)$. Although X_n decides which $(q,*)$[7] will be in Q_n^{Easy} and Q_n^{Hard}, the answer part $\mathsf{O}_n(q)$'s and $\mathsf{O}'_n(q)$'s are uniformly distributed, *independent* of X_n. That is, if O_n and O'_n are sampled randomly, then for any $X_n \in \{0,1\}^{\ell(n)}$, $\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})$ will also be a random oracle. Therefore, for *any* $\{X_n\}_{n\in\mathbb{N}}$ where $X_n \in \{0,1\}^{\ell(n)}$, the following holds

$$\left\{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})\right\}_{n\in\mathbb{N}} \overset{\text{i.d.}}{\equiv} \{\mathsf{O}''_n\}_{n\in\mathbb{N}},$$

where each O''_n is sampled uniformly from the space of functions from $\{0,1\}^n$ to $\{0,1\}^n$. Since it follows from Lem. 1 that $\{\mathsf{O}''_n\}_{n\in\mathbb{N}}$ is one-way with probability 1, so is $\left\{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})\right\}_{n\in\mathbb{N}}$. □

Claim 2 (together with our assumption) implies that, with probability 1 taken over the sampling of $\mathcal{O}$ and $\mathcal{O}'$:

- $G^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})} : \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)+1}$ is pseudo-random against all (unbounded) adversaries that make polynomially many queries to the oracle $\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})$; **and**
- $\Pi^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$ is a semi-honest zero-knowledge argument for the Meta-FBB functionality $\mathcal{F}_{\mathrm{ZK}}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}[G]$.

Let $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$ denote the following execution:

$$\langle P^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}(X_n, Y_n), V^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}(Y_n)\rangle.$$

Let $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})} = 1$ denote the event that the verifier accepts at the end of this execution. Then, it follows from Claim 2 and the completeness of $\Pi^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$ that:

$$\Pr_{\mathcal{O},\mathcal{O}'}\left[\begin{array}{l}\text{For sufficient large } n \in \mathbb{N}, \forall X_n \in \{0,1\}^{\ell(n)}, Y_n = G^{\mathsf{O}_n}(X_n), \\ \Pr\left[\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})} = 1\right] \geq 1 - \delta_c(n)\end{array}\right] = 1, \tag{3}$$

where the inner probability is taken over the random coins of the prover and the verifier in the execution $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$, and $\delta_c(n)$ is the completeness error.

Hybrid H_1. This hybrid is identical to the previous one, except that H_1 executes the protocol

$$\langle P^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}(X_n, Y_n), V^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}(Y_n)\rangle. \tag{4}$$

(Compared with the execution $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$ in H_0, the only difference is that H_1 remove Q_n^{Hard} from the *verifier's* oracle.)

As mentioned in the **Proof Sketch** of Thm. 3, we want to show that the verifier accepts in Execution 4 with probability close to that in the execution $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$. This is formalized as Claim 3.

Claim 3. *With probability 1 taken over the sampling of* $\mathcal{O}$ *and* $\mathcal{O}'$*, for sufficiently large* $n \in \mathbb{N}$*, it holds that* $\forall X_n \in \{0,1\}^{\ell(n)}$ *and* $Y_n = G^{\mathsf{O}_n}(X_n)$*,*

$$\Pr\left[\langle P^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}(X_n, Y_n), V^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}(Y_n)\rangle = 1\right] \geq \Pr\left[\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})} = 1\right] - \frac{1}{n}, \tag{5}$$

where the probabilities in the above inequality are taken over the random coins of the prover and the verifier during the corresponding executions.

[7] The symbol "$*$" denotes the wildcard that matches any answer to q.

Proof. First, we remark that the "with probability 1" part in this claim is to ensure that $\{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})\}_n$ is one-way (see Claim 2). In the following, we proceed with $\{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})\}_n$ being one-way (so the probabilities below are not taken over $\mathcal{O}$ and $\mathcal{O}'$).

By definition, any query[8] $q \in Q_n^{\mathsf{Hard}}$ is asked by V during $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$ with probability $< \frac{1}{t(n)\cdot n}$. Let us define the following event:

- $\mathsf{Event}_{\mathsf{NoHard}}$: no $q \in Q_n^{\mathsf{Hard}}$ is asked by V in $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$.

It follows from the union bound that

$$\Pr\left[\mathsf{Event}_{\mathsf{NoHard}}\right] \geq 1 - \frac{1}{n}, \tag{6}$$

where the probability is taken over the random coins of P and V in the execution $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$.

Now, we prove Inequality (5). In the following, for succinctness, let

- Exec_0 denote the execution $\langle P^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}(X_n, Y_n), V^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}(Y_n)\rangle$;
- Exec_1 denote the execution $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$.

Then, we have (probabilities below are taken over the random coins over P and V in the corresponding executions):

$$\begin{aligned}
\Pr[\mathsf{Exec}_1 = 1] &\geq \Pr\left[\mathsf{Exec}_1 = 1 \,\middle|\, \mathsf{Event}_{\mathsf{NoHard}}\right] \cdot \Pr\left[\mathsf{Event}_{\mathsf{NoHard}}\right] \\
&= \Pr\left[\mathsf{Exec}_0 = 1 \,\middle|\, \mathsf{Event}_{\mathsf{NoHard}}\right] \cdot \Pr\left[\mathsf{Event}_{\mathsf{NoHard}}\right] && (7)\\
&\geq \Pr[\mathsf{Exec}_0 = 1] - \Pr[\neg \mathsf{Event}_{\mathsf{NoHard}}] && (8)\\
&\geq \Pr[\mathsf{Exec}_0 = 1] - \frac{1}{n} && (9)
\end{aligned}$$

where Step 7 is due to the fact that Exec_1 and Exec_0 are identical assuming V does not make any query $q \in Q_n^{\mathsf{Hard}}$, Step 8 follows from the basic probability inequality that $\Pr\left[A \,\middle|\, B\right] \cdot \Pr[B] \geq \Pr[A] - \Pr[\neg B]$, and Step 9 follows from Inequality (6).

This finishes the proof of Claim 3. □

Claim 3 indicates that the verifier in Execution 4 accepts with "good" probability: at least as large as the accepting probability of $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n \diamond (Q_n^{\mathsf{Easy}} \cup Q_n^{\mathsf{Hard}})}$ minus $1/n$. Thus, we will have the desired contradiction if the Y_n in Execution 4 is a false statement, namely that Y_n is not in the range of $G^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}$ (i.e. $G^{(\cdot)}$ instantiated by the verifier's oracle in Execution 4). This argument is formalized and proved in Claims 4 and 5, which will eventually finish the proof of Thm. 3.

Claim 4. *Let Q_n^{Easy} be defined as in Expression (2). For sufficiently large $n \in \mathbb{N}$, the following holds:*

$$\Pr_{\mathcal{O},\mathcal{O}',X_n}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \leq \frac{1}{2} + \mathsf{negl}(n). \tag{10}$$

Note that the above probability is taken (additionally) over $X_n \leftarrow \{0,1\}^{\ell(n)}$.

Claim 5. *If Claim 4 holds, then Thm. 3 holds.*

The proof of Claim 4 is quite involved. It constitutes the main technical challenge of the current proof (of Thm. 3). Thus, we will deal with it in Sec. 4.4. In the following, we show the proof of Claim 5.

[8] Technically, elements in Q_n^{Hard} are query-answer pairs. From here on, we override the notation "$\in$" such that $q \in Q_n^{\mathsf{Hard}}$ also means that there exists a pair $(q, *)$ in Q_n^{Hard}.

Proof of Claim 5. It follows from Expression (3) and Claim 3 that

$$\Pr_{\mathcal{O},\mathcal{O}'}\left[\begin{array}{l}\text{For sufficient large } n\in\mathbb{N}, \forall X_n\in\{0,1\}^{\ell(n)}, Y_n=G^{\mathsf{O}_n}(X_n),\\ \Pr\left[\langle P^{\mathsf{O}'_n\diamond(Q_n^{\mathsf{Easy}}\cup Q_n^{\mathsf{Hard}})}(X_n,Y_n), V^{\mathsf{O}'_n\diamond Q_n^{\mathsf{Easy}}}(Y_n)\rangle=1\right]\geq 1-\frac{1}{n}-\delta_c(n)\end{array}\right]=1. \tag{11}$$

Following the same argument as for Claim 2, we can prove the one-wayness of the oracle $\{\mathsf{O}'_n\diamond Q_n^{\mathsf{Easy}}\}_n$ as follows. For each $(q,\mathsf{O}_n(q))\in Q_n^{\mathsf{Easy}}$, the $\mathsf{O}_n(q)$ is a randomly sampled string from $\{0,1\}^n$. Therefore, no matter what X_n is, $\mathsf{O}'_n\diamond Q_n^{\mathsf{Easy}}$ is always a randomly sampled oracle (though Q_n^{Easy} is determined by X_n). It then follows from Lem. 1 that:

$$\Pr_{\mathcal{O},\mathcal{O}'}\left[\forall X_n\in\{0,1\}^{\ell(n)},\ \{\mathsf{O}'_n\diamond Q_n^{\mathsf{Easy}}\}_{n\in\mathbb{N}}\text{ is one-way}\right]=1. \tag{12}$$

By an averaging argument over Expressions (10) to (12), it follows that there exists fixed sequences $\{\ddot{\mathsf{O}}_n\}_{n\in\mathbb{N}}$, $\{\ddot{\mathsf{O}}'_n\}_{n\in\mathbb{N}}$ and $\{\ddot{X}_n\}_{n\in\mathbb{N}}$[9] such that for sufficiently large $n\in\mathbb{N}$,

- $\{\ddot{\mathsf{O}}_n\diamond\ddot{Q}_n^{\mathsf{Easy}}\}_n$ is one-way; thus, $G^{\ddot{\mathsf{O}}_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$ is a PRG and $\Pi^{\ddot{\mathsf{O}}_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$ is an HVZK protocol for the membership of $G^{\ddot{\mathsf{O}}_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$; **and**
- $\ddot{Y}_n$ is not in the range of $G^{\ddot{\mathsf{O}}_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$; **and**
- $\Pr\left[\langle P^{\ddot{\mathsf{O}}'_n\diamond(\ddot{Q}_n^{\mathsf{Easy}}\cup\ddot{Q}_n^{\mathsf{Hard}})}(\ddot{X}_n,\ddot{Y}_n), V^{\ddot{\mathsf{O}}'_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}(\ddot{Y}_n)\rangle=1\right]\geq 1-\frac{1}{n}-\delta_c(n)$, where the probability is taken over the random coins of P and V.
  Note that we can treat $P^{\ddot{\mathsf{O}}'_n\diamond(\ddot{Q}_n^{\mathsf{Easy}}\cup\ddot{Q}_n^{\mathsf{Hard}})}(\ddot{X}_n,\ddot{Y}_n)$ as an oracle machine $\mathbb{P}^{\ddot{\mathsf{O}}'_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$, which has $(\ddot{Q}_n^{\mathsf{Easy}},\ddot{X}_n,\ddot{Y}_n)$ as non-uniform advice and makes only polynomially many queries to its oracle $\ddot{\mathsf{O}}'_n\diamond\ddot{Q}_n^{\mathsf{Easy}}$.

Since the completeness error $\delta_c(\cdot)$ is negligible, the above means that $\mathbb{P}^{\ddot{\mathsf{O}}'_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$ (with its non-uniform advice) convinces the verifier with non-negligible probability on the following *false* statement:

$$\ddot{Y}_n\in G^{\ddot{\mathsf{O}}'_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right).$$

This contradicts the soundness of $\Pi^{\ddot{\mathsf{O}}'_n\diamond\ddot{Q}_n^{\mathsf{Easy}}}$, thus finishing the proof of Claim 5. □

4.4 Proof of Claim 4

Proof Overview. Before diving into the proof, we first provide a high-level overview.

We assume for contradiction that Claim 4 is false and try to break the pseudo-randomness of G^{O_n}. First, observe that if $Y_n=G^{\mathsf{O}_n}(X_n)$ where $X_n\leftarrow\{0,1\}^{\ell(n)}$, then our assumption implies that Y_n is in the range of $G^{\mathsf{O}'_n\diamond Q_n^{\mathsf{Easy}}}(\cdot)$ with probability noticeably larger than $1/2$. Therefore, on an input Y_n, if we can efficiently test if $Y_n\in G^{\mathsf{O}'_n\diamond Q_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)$, we should have some advantage in the PRG game for $G^{\mathsf{O}_n}(\cdot)$. This strategy has the following potential problems:

1. Without the preimage X_n, we *cannot* compute the set Q_n^{Easy} (see Expression (2)) using only polynomially many queries to O_n;
2. If the input $Y_n\notin G^{\mathsf{O}_n}\left(\{0,1\}^{\ell(n)}\right)$, the set Q_n (thus Q_n^{Easy}) is not even well-defined, as there is no preimage X_n.

To avoid using X_n, we will run the HVZK simulator to obtain an estimate of the set Q_n^{Easy} in the following way. Recall that Q_n^{Easy} contains the "easy" queries made by the verifier during $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}'_n\diamond Q_n}$. By the HVZK property of the protocol $\Pi^{\mathsf{O}'_n\diamond Q_n}$, each query in Q_n^{Easy} should be made with similar probability in the simulated execution $\mathsf{Sim}_V^{\mathsf{O}'_n\diamond Q_n}(Y_n)$. Therefore, repeating $\mathsf{Sim}_V^{\mathsf{O}'_n\diamond Q_n}(Y_n)$ (polynomially) many times will give us a good estimate to Q_n^{Easy}.

[9] Note that these values also fix the corresponding $\{\ddot{Y}_n\}_{n\in\mathbb{N}}$, $\{\ddot{Q}_n^{\mathsf{Easy}}\}_{n\in\mathbb{N}}$ and $\{\ddot{Q}_n^{\mathsf{Hard}}\}_{n\in\mathbb{N}}$ as in the above.

Algorithm 1: Sampling the Set $\widetilde{Q}_n^{\mathsf{Easy}}$

Input: a string $Y_n \in \{0,1\}^{\ell(n)+1}$.

Oracle: an oracle O_n mapping $\{0,1\}^n$ to $\{0,1\}^n$.

Let $0 < c < 1$ be a constant. This algorithm initializes a table $\mathcal{T}$ to the records of the form $\big(q_i, \mathsf{Count}[q_i] = 0\big)$ for all $q_i \in \{0,1\}^n$. $\mathcal{T}$ will be used to store the number that each query being asked.

This algorithm repeats the following procedure for $N = 3n/c^2$ times, using fresh randomness for each repetition:

- It invokes the HVZK simulator $\mathsf{Sim}_V^{\mathsf{O}_n}(Y_n)$. Note that the simulated view $\mathsf{View} \leftarrow \mathsf{Sim}_V^{\mathsf{O}_n}(Y_n)$ contains the query-answer pairs exchanged between V and O_n. For every query-answer pair $(q_i, \mathsf{O}_n(q_i))$ appeared in View, increase $\mathsf{Count}[q_i]$ by 1.

For any query $q_i \in \{0,1\}^n$, let $\widehat{p}_i$ denotes the frequency (i.e. the empirical mean) that q_i is asked by the verifier during the above N repetitions. The hybrid then computes the set $\widetilde{Q}_n^{\mathsf{Easy}}$ as follows:

$$\widetilde{Q}_n^{\mathsf{Easy}} := \Big\{ \big(q_i, \mathsf{O}_n(q_i)\big) \;\Big|\; \widehat{p}_i \geq \frac{1}{n \cdot t(n)} - 2c \Big\}, \quad \text{where } \widehat{p}_i := \frac{\mathsf{Count}[q_i]}{N}. \tag{14}$$

However, without X_n, we cannot figure out the set Q_n, which is necessary if we want to run $\mathsf{Sim}_V^{\mathsf{O}'_n \diamond Q_n}(Y_n)$. Fortunately, by a similar argument as that for Claim 2, we can prove that the oracle $\{\mathsf{O}_n\}_n$ and $\{\mathsf{O}'_n \diamond Q_n\}_n$ are identically distributed, even given X_n and $Y_n = G^{\mathsf{O}_n}(X_n)$. Therefore, running $\mathsf{Sim}_V^{\mathsf{O}_n}(Y_n)$ will be just as good as running $\mathsf{Sim}_V^{\mathsf{O}'_n \diamond Q_n}(Y_n)$. Note that this also solves Problem 2, because the simulator still works when invoked on false statements.

Now, we can construct the PRG distinguisher $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}(Y_n)$ as follows: on input Y_n, $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}(Y_n)$ obtains an estimate $\widetilde{Q}_n^{\mathsf{Easy}}$ to Q_n^{Easy} by running $\mathsf{Sim}_V^{\mathsf{O}_n}(Y_n)$ polynomially many times. It then samples a random function $\mathsf{O}'_n : \{0,1\}^n \to \{0,1\}^n$, and outputs 1 if $Y_n \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\big(\{0,1\}^{\ell(n)}\big)$; otherwise, it outputs 0. Note that although sampling O' requires exponential time, $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}(Y_n)$ only makes *polynomially many* queries to the oracle O_n.

If $Y_n = G^{\mathsf{O}_n}(X_n)$ where $X_n \leftarrow \{0,1\}^{\ell(n)}$, then by our assumption $\mathcal{A}^{\mathsf{O}_n}(Y_n)$ outputs 1 with probability noticeably larger than $1/2$; if $Y_n \leftarrow \{0,1\}^{\ell(n)+1}$, then Y_n is independent of O_n. Moreover, using a similar argument as for Claim 2, we can prove that Y_n is independent of the oracle $\widetilde{Q}_n^{\mathsf{Easy}}$ (thus $\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}$). Since the function $G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}(\cdot)$ stretch by 1 bit, the random Y_n will be in its range with probability $1/2$. This means $\mathcal{A}^{\mathsf{O}_n}(Y_n)$ outputs 1 with probability exactly $1/2$.

This gives us the desired contradiction.

The Formal Proof. We now present the formal proof for Claim 4. First, we describe in Algo. 1 how to compute the set $\widetilde{Q}_n^{\mathsf{Easy}}$, which is the estimate to Q_n^{Easy} by running $\mathsf{Sim}_V^{\mathsf{O}_n}(Y_n)$ (see the above proof overview). In the following, we break Claim 4 into Claims 6 and 8, and prove them one-by-one.

Claim 6. *For sufficiently large $n \in \mathbb{N}$, it holds that*

$$\Pr_{\mathcal{O},\mathcal{O}',X_n}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}\big(\{0,1\}^{\ell(n)}\big)\right] \leq \Pr_{\mathcal{O},\mathcal{O}',X_n,\widetilde{Q}_n^{\mathsf{Easy}}}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\big(\{0,1\}^{\ell(n)}\big)\right] + \mathsf{negl}(n). \tag{13}$$

Proof. Let $Y_n = G^{\mathsf{O}_n}(X_n)$. Let $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}_n}$ denote the execution $\langle P^{\mathsf{O}_n}(X_n,Y_n), V^{\mathsf{O}_n}(Y_n)\rangle$. For each $q_i \in \{0,1\}^n$, let p_i denote the probability that q_i is asked *by the verifier* during $\mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}_n}$. We now define the following set:

$$\bar{Q}_n^{\mathsf{Easy}} := \Big\{ \big(q_i, \mathsf{O}_n(q_i)\big) \;\Big|\; p_i \geq \frac{1}{t(n) \cdot n} \text{ during } \mathsf{Exec}_{X_n,Y_n}^{\mathsf{O}_n} \Big\}. \tag{15}$$

We remark that $\bar{Q}_n^{\mathsf{Easy}}$ is the same as Q_n^{Easy} (Expression (2)) but is w.r.t. $\mathsf{Exec}^{\mathsf{O}_n}_{X_n,Y_n}$. In Claim 7, we show that $\bar{Q}_n^{\mathsf{Easy}}$ and Q_n^{Easy} are identically distributed. Looking ahead, this claim will allow us to replace Q_n^{Easy} with $\bar{Q}_n^{\mathsf{Easy}}$, for which we can obtain a "good" estimate (via the HVZK simulator) without knowing the preimage X_n.

Claim 7. *Let Q^{Easy} be as in Expression (2). Let $\bar{Q}^{\mathsf{Easy}}$ be as in Expression (15). The following holds:*

$$\Pr_{\mathcal{O},\mathcal{O}',X_n}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond Q_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] = \Pr_{\mathcal{O},\mathcal{O}',X_n}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \bar{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \tag{16}$$

Proof. We first claim that the following two ensembles are identically distributed:

$$\{X_n, G^{\mathsf{O}_n}(X_n), \mathsf{O}_n\}_{n\in\mathbb{N}} \overset{\text{i.d.}}{\equiv} \{X_n, G^{\mathsf{O}_n}(X_n), \mathsf{O}'_n \diamond Q_n\}_{n\in\mathbb{N}}, \tag{17}$$

where $X_n \leftarrow \{0,1\}^{\ell(n)}$, O_n and O'_n are random functions mapping $\{0,1\}^n$ to $\{0,1\}^n$, and Q_n is defined as in hybrid H_0 (the set of query-answer pairs during the evaluation of $G^{\mathsf{O}_n}(X_n)$). To show Expression (17), we only need to show that given $\{X_n\}_n$, the oracles $\{\mathsf{O}_n\}_n$ and $\{\mathsf{O}'_n \diamond Q_n\}_n$ are identically distributed. This is true as they agree on all the query-answer pairs contained in Q_n, and the queries (and their answers) not in Q_n are identically distributed (uniform).

Then, note that $\{\bar{Q}^{\mathsf{Easy}}\}_n$ is determined (deterministically) by the left-hand side of Expression (17) *in the same way* as $\{Q\}_n$ is determined by the right-hand side of Expression (17). So $\{Q^{\mathsf{Easy}}\}_n$ and $\{\bar{Q}^{\mathsf{Easy}}\}_n$ are also identically distributed. Thus, Eq. (16) follows. □

Next, we prove that the $\widetilde{Q}_n^{\mathsf{Easy}}$ defined in Algo. 1 is a good estimate to the set $\bar{Q}_n^{\mathsf{Easy}}$ in the sense that $\bar{Q}_n^{\mathsf{Easy}}$ is a subset of $\widetilde{Q}_n^{\mathsf{Easy}}$ except with negligible probability taken over the sampling of $\widetilde{Q}_n^{\mathsf{Easy}}$. For any $q_i \in \{0,1\}^n$, let $\widetilde{p}_i$ denote the probability that q_i is asked by the verifier in the simulated transcript $\mathsf{View} \leftarrow \mathsf{Sim}_V^{\mathsf{O}_n}(Y_n)$. Due to the HVZK property of Π^{O_n}, it holds that $\widetilde{p}_i \geq p_i - \mathsf{negl}(n)$. Recall the quantity $\widehat{p}_i$ from Expression (14), which is the empirical mean of the frequency that p_i is asked by the verifier in $3n/c^2$ repetitions of $\mathsf{Sim}_V^{\mathsf{O}_n}(Y)$. It then follows from the Chernoff bound that

$$\Pr_{\widehat{p}_i}\left[|\widehat{p}_i - \widetilde{p}_i| \geq c\right] \leq \frac{1}{2^n} = \mathsf{negl}(n). \tag{18}$$

By definition, each $q_i \in \bar{Q}_n^{\mathsf{Easy}}$ will be asked with probability $p_i \geq \frac{1}{n\cdot t(n)}$ during the execution $\mathsf{Exec}^{\mathsf{O}_n}_{X_n,Y_n}$. It then follows from Inequality (18) that for any $q_i \in \bar{Q}_n^{\mathsf{Easy}}$, the following holds except with negligible probability taken over the sampling of $\widehat{p}_i$:

$$\widehat{p}_i \geq \widetilde{p}_i - c = p_i - c - \mathsf{negl}(n) \geq \frac{1}{n\cdot t(n)} - c - \mathsf{negl}(n) > \frac{1}{n\cdot t(n)} - 2c,$$

which means $q_i \in \widetilde{Q}_n^{\mathsf{Easy}}$ by the definition of $\widetilde{Q}_n^{\mathsf{Easy}}$. Since $|\bar{Q}_n^{\mathsf{Easy}}|$ is a polynomial of n, it follows from union bound that

$$\Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}\left[\bar{Q}_n^{\mathsf{Easy}} \subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \geq 1 - \mathsf{negl}(n) \quad \left(\Leftrightarrow \Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}\left[\bar{Q}_n^{\mathsf{Easy}} \not\subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \leq \mathsf{negl}(n)\right). \tag{19}$$

Then we have

$$\begin{aligned}
&\Pr_{\mathcal{O},\mathcal{O}',X_n}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \bar{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \\
&\leq \Pr\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \bar{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right) \mid \bar{Q}_n^{\mathsf{Easy}} \subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \cdot \Pr\left[\bar{Q}_n^{\mathsf{Easy}} \subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] + \Pr\left[\bar{Q}_n^{\mathsf{Easy}} \not\subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \\
&\leq \Pr\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right) \mid \bar{Q}_n^{\mathsf{Easy}} \subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \cdot \Pr\left[\bar{Q}_n^{\mathsf{Easy}} \subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] + \Pr\left[\bar{Q}_n^{\mathsf{Easy}} \not\subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \\
&\leq \Pr\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] + \Pr\left[\bar{Q}_n^{\mathsf{Easy}} \not\subseteq \widetilde{Q}_n^{\mathsf{Easy}}\right] \\
&\leq \Pr_{\mathcal{O},\mathcal{O}',X_n,\widetilde{Q}_n^{\mathsf{Easy}}}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] + \mathsf{negl}(n)
\end{aligned} \tag{20}$$

where Inequality (20) follows from Inequality (19).

Eq. (16) and Inequality (20) finish the proof of Claim 6. □

Remark 3 (On the Probability Space). Algo. 1 run the HVZK simulator for the protocol Π^{O_n}. We want to point out that Π^{O_n} is an HVZK protocol (i.e. the simulator exists) only if $\mathcal{O} = \{\mathsf{O}_n\}_n$ is one-way. Therefore, in the above proof, whenever we try to argue about some probability of the form

$$\Pr_{\mathcal{O},\widetilde{Q}_n^{\mathsf{Easy}}}\left[\mathsf{Event}(\mathsf{O}_n, \widetilde{Q}_n^{\mathsf{Easy}})\right] = \mathsf{value} \tag{21}$$

with the probability taken over *both* $\mathcal{O}$ and $\widetilde{Q}_n^{\mathsf{Easy}}$ (e.g. Inequalities (18), (19) and (20), and some expressions in the next claim), the technically correct way is to say: with probability 1 taken over $\mathcal{O} = \{\mathsf{O}_n\}_n$ (thus being one-way), it holds that

$$\Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}\left[\mathsf{Event}(\mathsf{O}_n, \widetilde{Q}_n^{\mathsf{Easy}})\right] = \mathsf{value}. \tag{22}$$

Or, put in another way:

1. $\Pr_{\mathcal{O}}[\mathcal{O} \text{ is one-way}] = 1$; **and**
2. $\Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}\left[\mathsf{Event}(\mathsf{O}_n, \widetilde{Q}_n^{\mathsf{Easy}}) \;\middle|\; \mathcal{O} \text{ is one-way}\right] = \mathsf{value}$.

However, the form of Expression (21) is also correct because it is actually a consequence of Expression (22): let A denote the event $\mathsf{Event}(\mathsf{O}_n, \widetilde{Q}_n^{\mathsf{Easy}})$, and B the event that $\mathcal{O}$ is one-way, then

$$\begin{aligned}\Pr_{\mathcal{O},\widetilde{Q}_n^{\mathsf{Easy}}}[A] &= \Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}[A \mid B] \cdot \Pr_{\mathcal{O}}[B] + \Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}[A \mid \neg B] \cdot \Pr_{\mathcal{O}}[\neg B] \\ &= \Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}[A \mid B] \cdot 1 + \Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}[A \mid \neg B] \cdot 0 = \Pr_{\widetilde{Q}_n^{\mathsf{Easy}}}[A \mid B] = \mathsf{value}.\end{aligned}$$

So, it is fine to use Expression (21).

Claim 8. *For sufficiently large $n \in \mathbb{N}$, it holds that*

$$\Pr_{\mathcal{O},\mathcal{O}',X_n,\widetilde{Q}_n^{\mathsf{Easy}}}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \leq \frac{1}{2} + \mathsf{negl}(n) \tag{23}$$

Proof. At a high level, this proof goes as follows. Assuming for contradiction that the claim does not hold, we show an adversary $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ that breaks the pseudo-randomness of $G^{\mathsf{O}_n}(\cdot)$. This gives us the desired contradiction, because $G^{\mathsf{O}_n}(\cdot)$ is a PRG given that $\mathcal{O} = \{\mathsf{O}_n\}_{n\in\mathbb{N}}$ is one-way.

Formally, we assume for contradiction that there exists a polynomial $\mathsf{poly}_{\mathrm{PRG}}(\cdot)$ such that, for infinitely many $n \in \mathbb{N}$, the following holds:

$$\Pr_{\mathcal{O},\mathcal{O}',X_n,\widetilde{Q}_n^{\mathsf{Easy}}}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_h^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \geq \frac{1}{2} + \frac{1}{\mathsf{poly}_{\mathrm{PRG}}(n)}. \tag{24}$$

On input $Y_n \in \{0,1\}^{\ell(n)+1}$, $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ proceeds as follows:

1. Compute $\widetilde{Q}_n^{\mathsf{Easy}}$ using Algo. 1;
2. $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ samples O'_n uniformly from all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$. With the $\widetilde{Q}_n^{\mathsf{Easy}}$ from Step 1, $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ now has the full description of $\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}$. $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ then tests if $Y \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)$. We remark that it takes exponential computation to sample O'_n. However, this step does *not* incur any calls to the oracle O_n.
3. **Output:** If $Y_n \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)$, output 1; otherwise, output 0.

If Y_n is the output of $G^{\mathsf{O}_n}(\cdot)$ on a random X_n, then it follows from above description of $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ that

$$\begin{aligned}\Pr_{\mathcal{O},\mathcal{O}',X_n}\left[\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}\left(G^{\mathsf{O}_n}(X_n)\right) = 1\right] &= \Pr_{\mathcal{O},\mathcal{O}',X_n,\widetilde{Q}_n^{\mathsf{Easy}}}\left[G^{\mathsf{O}_n}(X_n) \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \\ &\geq \frac{1}{2} + \mathsf{negl}(n);\end{aligned} \tag{25}$$

If $Y_n \leftarrow \{0,1\}^{\ell(n)+1}$, then the oracle $\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}$ and Y_n are independently distributed. We emphasize that although $\widetilde{Q}_n^{\mathsf{Easy}}$ is obtained by running $\mathsf{Sim}_V^{\mathsf{O}_n}$ on Y_n (for $N = 3n/c^2$ times), for each $(q, \mathsf{O}_n(q)) \in \widetilde{Q}_n^{\mathsf{Easy}}$, the answer $\mathsf{O}_n(q)$ is independent of Y_n. Therefore, Y_n is independent of $\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}$. In this case, Y_n is the range of the PRG with probability exactly $1/2$. Formally,

$$\begin{aligned}\Pr_{\mathcal{O},\mathcal{O}',Y_n}\left[\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}(Y_n)=1\right] &= \Pr_{\mathcal{O},\mathcal{O}',Y_n,\widetilde{Q}_n^{\mathsf{Easy}}}\left[Y_n \in G^{\mathsf{O}'_n \diamond \widetilde{Q}_n^{\mathsf{Easy}}}\left(\{0,1\}^{\ell(n)}\right)\right] \\ &= \Pr_{Y_n,\mathsf{O}''_n}\left[Y_n \in G^{\mathsf{O}''_n}\left(\{0,1\}^{\ell(n)}\right)\right] = \frac{1}{2} \end{aligned} \tag{26}$$

Expressions (25) and (26) imply that $\mathcal{A}_{\mathrm{PRG}}^{\mathsf{O}_n}$ breaks the pseudo-randomness of $G^{\mathsf{O}_n}(\cdot)$. This completes the proof of Claim 8. □

This completes the proof of Claim 4.

5 Proof-Based One-Way Functions

5.1 Definition

Definition 12 (Proof-Based OWFs). *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $a(\cdot)$, $b(\cdot)$ and $c(\cdot)$ be polynomials. A* proof-based one-way function *consists of a function $F_\lambda : \{0,1\}^{a(\lambda)} \times \{0,1\}^{b(\lambda)} \to \{0,1\}^{c(\lambda)}$ and a protocol $\Pi = (S, R)$ of a pair of* PPT *machines. We use $(X, Y) \leftarrow \langle S(1^\lambda, \mathsf{x}), R(1^\lambda, \mathsf{r})\rangle$ to denote the execution of protocol Π where the security parameter is λ, the inputs to S and R are x and r respectively, and the outputs of S and R are X and Y respectively. Let $Y = \perp$ denote that R aborts in the execution. The following conditions hold:*

- **One-Wayness.** *The function $\{F_\lambda\}_\lambda$ is one-way in the following sense:*
  - Easy to compute: *for all $\lambda \in \mathbb{N}$ and all $(\mathsf{x}, \mathsf{r}) \in \{0,1\}^{a(\lambda)} \times \{0,1\}^{b(\lambda)}$, $F_\lambda(\mathsf{x}\|\mathsf{r})$ can be computed in polynomial time on λ.*
  - Hard to invert: *for any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$,*

$$\Pr\left[\mathsf{x} \leftarrow \{0,1\}^{a(\lambda)}, X^* \leftarrow \mathcal{A}\left(1^\lambda, F_\lambda(\mathsf{x}\|\mathsf{r})\right) : F_\lambda(\mathsf{x}\|\mathsf{r}) = F_\lambda(X^*)\right] \leq \mathsf{negl}(\lambda),$$

- **Completeness.** *The protocol Π computes the ideal functionality $\mathcal{F}_F$ defined in Fig. 1. Namely, $\forall \lambda \in \mathbb{N}$, $\forall \mathsf{x} \in \{0,1\}^{a(\lambda)}$ and $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$, if $(X, Y) \leftarrow \langle S(1^\lambda, \mathsf{x}), R(1^\lambda, \mathsf{r})\rangle$, then $X = \mathsf{x}\|\mathsf{r}$ and $Y = F_\lambda(\mathsf{x}\|\mathsf{r})$.*
- **Soundness.** *For every* PPT *machine S^* and every auxiliary input $z \in \{0,1\}^*$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\begin{array}{l|l} \mathsf{r} \leftarrow \{0,1\}^{b(\lambda)}; & Y \neq \perp \textbf{ and} \\ (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle & \nexists \mathsf{x} \text{ s.t. } F_\lambda(\mathsf{x}\|\mathsf{r}) = Y \end{array}\right] \leq \mathsf{negl}(\lambda),$$

- **Zero-Knowledge.** *This property is defined by requiring* only *security against corrupted R in the ideal-real paradigm for 2PC w.r.t. the ideal functionality $\mathcal{F}_F$ in Fig. 1. Concretely, denote by $\mathsf{REAL}_{\Pi,\mathcal{A}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})$ the random variable consisting of the output of S and the output of the adversary $\mathcal{A}$ controlling R in an execution of Π, where x is the input to S and r to R. Similarly, denote by $\mathsf{IDEAL}_{\mathcal{F}_F,\mathsf{Sim}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})$ the corresponding output of S and Sim from the ideal execution.[10] Then there exist a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$, $\forall \mathsf{x} \in \{0,1\}^{a(\lambda)}$, $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$, and $\forall z \in \{0,1\}^*$, $\left\{\mathsf{REAL}_{\Pi,\mathcal{A}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})\right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{IDEAL}_{\mathcal{F}_F,\mathsf{Sim}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})\right\}_{\lambda\in\mathbb{N}}$.*

If the constructions of both F and Π makes only black-box access to other primitives, we call this a black-box PB-OWF.

[10] We refer the reader to [Gol04] for a detailed description of the ideal and real executions.

Figure 1: Functionality $\mathcal{F}_F$ for Proof-Based OWFs

The ideal functionality $\mathcal{F}_F$ interacts with a sender S and a receiver R. Upon receiving the input $\mathsf{x} \in \{0,1\}^{a(\lambda)}$ from S and $\mathsf{r} \in \{0,1\}^{b(\lambda)}$ from R, the functionality $\mathcal{F}_F$ sends $\mathsf{x}\|\mathsf{r}$ to S, and $F(\mathsf{x}\|\mathsf{r})$ to R.

Construction 1: One-Way Function F^f

Let $m(\lambda)$ and $n(\lambda)$ be polynomials on λ. Let $0 < \delta < 1$ be a constant, and $k(\lambda) = \delta n(\lambda)$. Let $t(\lambda) = \log^2(\lambda)$ (see Rmk. 4). Assume that $f : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ is a one-way function. On input $\mathsf{x} \in \{0,1\}^{n\lambda+(\log(n)+m)k}$ and $\mathsf{r} \in \{0,1\}^{t\log(n)}$, F^f parses them as

$$\mathsf{x} = (x_1, \ldots, x_n)\|(p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k}), \text{ and } \mathsf{r} = (b_1, \ldots, b_t),$$

where $|x_i| = \lambda$, $|y'_{p_i}| = m$, $\{p_i\}_{i\in[k]}$ is a size-k subset of $[n]$, and $\{b_i\}_{i\in[t]}$ is a size-t subset of $[n]$. F^f computes via its oracle access to $f(\cdot)$ the values $(y_1, \ldots, y_n)$, where $y_i = f(x_i)$ for all $i \in [n]$. Then, it computes $\mathsf{s} = (s_1, \ldots, s_n)$ as follows:

1. if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then let $s_i \coloneqq y_i$ for all $i \in [n]$.
2. if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then let $s_i \coloneqq \begin{cases} y'_i & i \in \{p_1, \ldots, p_k\} \\ y_i & i \in [n] \setminus \{p_1, \ldots, p_k\} \end{cases}$.

It finally outputs $Y = (s_1, \ldots, s_n)\|(x_{b_1}, \ldots, x_{b_t})\|(b_1, \ldots, b_t)$.

5.2 Our Construction

Following the high-level idea described in Sec. 2.2, we show that PB-OWFs can be obtained assuming black-box access to OWFs.

Theorem 4 (Black-Box PB-OWFs from OWFs). *There exists a PB-OWF that satisfies Def. 12 and makes only black-box use of OWFs.*

Our construction consists of a one-way function F^f (Constr. 1) together with a protocol Π_F^f (Prot. 1). The construction relies on the following building blocks:

- a one-way function f;
- a zero-knowledge commit-and-prove protocol $\Pi_{\mathsf{ZKCnP}} = (\mathsf{BBCom}, \mathsf{BBProve})$ as per Def. 8. Such protocols can also be constructed assuming only black-box access to f.

It follows immediately from the description that our construction makes only black-box access to OWFs.

Remark 4 (On the Parameters in Constr. 1). The choice of $t(\lambda) = \log^2(\lambda)$ is somewhat arbitrary. In fact, any $t(\lambda) = \omega(\log \lambda)$ works as long as $(n - k - t)$ is some positive polynomial of λ for sufficiently large λ. This is to ensure that we can prove one-wayness in Lem. 2 *and* $(1 - \delta)^t$ is negligible on λ, which is needed when we prove soundness (Claim 9). We also remark that the role of r is to specify a size-t subset of $[n]$. The canonical way of mapping r to a size-t subset of $[n]$ may consume slightly less randomness than $|\mathsf{r}| = t\log(n)$. For simplicity, we forego further discussion and assume that there is a deterministic bijection between $\{0,1\}^{t\log(n)}$ and all size-t subsets of $[n]$. Similarly, the $\{p_1, \ldots, p_k\}$ are interpreted as a size-k subset of $[n]$, though we assign each p_i a length of $\log(n)$.

5.3 Security Proof for Our PB-OWFs

Proof Overview. Before presenting the formal proof of security, we first provide an overview.

One-wayness, completeness, and ZK follow from rather standard techniques. In the following, let us explain more about the soundness proof (shown formally as Lem. 3).

Protocol 1: Protocol Π_F^f for Our Proof-Based One-Way Function

Let f, m, n, t, and k be as in Constr. 1.

Input: the security parameter 1^λ is the common input. Sender S takes $\mathsf{x} \in \{0,1\}^{n\lambda+(\log(n)+m)k}$ as its private input; receiver R takes $\mathsf{r} \in \{0,1\}^{t\cdot\log(n)}$ as its private input.

1. S parses the input as $\mathsf{x} = (x_1, \ldots, x_n) \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k})$, where $|x_i| = \lambda$ for all $i \in [n]$, $|y'_{p_j}| = m$ for all $j \in [k]$, and $\{p_i\}_{i\in[k]}$ forms a size-k subset of $[n]$. S defines a $2 \times n$ matrix $M = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}$, where $y_i = f(x_i)$ for all $i \in [n]$.
2. S and R execute $\mathsf{BBCom}(\alpha)$, the **Commit** stage of Π_{ZKCnP}, where S commits to the value
$$\alpha := M \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k}). \tag{27}$$
3. R sends r to S.
4. S interprets r as a size-t subset $(b_1, \ldots, b_t) \subseteq [n]$. S then defines $M_\mathsf{r} = \begin{bmatrix} x_{b_1} & \cdots & x_{b_t} \\ y_{b_1} & \cdots & y_{b_t} \end{bmatrix}$, i.e. the columns of M specified by r. S also computes $\mathsf{s} = (s_1, \ldots, s_n)$ in the way specified in Constr. 1. S sends to R the values M_r and s.
5. With M_r, R checks (via its oracle access to $f(\cdot)$) if $f(x_{b_i}) = y_{b_i}$ holds for all $i \in [t]$; R also checks if $s_{b_i} = y_{b_i}$ holds for all $i \in [t]$. If all the checks pass, R proceeds to next step; otherwise, R halts and outputs $\perp$.
6. S and R execute $\mathsf{BBProve}$, the **Prove** stage of Π_{ZKCnP}, where S proves that it performs Stage 4 honestly. Namely, S proves that the α committed at Stage 2 satisfies the following conditions:
   (a) the values $\{p_1, \ldots, p_k\}$ contained in α form a size-k subset of $[n]$; **and**
   (b) the M_r does consist of the columns in M specified by r; **and**
   (c) The $\mathsf{s} = (s_1, \ldots, s_n)$ satisfies the following conditions:
   - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then $s_i = y_i$ for all $i \in [n]$.
   - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then $s_i = \begin{cases} y'_i & i \in \{p_1, \ldots, p_k\} \\ y_i & i \in [n] \setminus \{p_1, \ldots, p_k\} \end{cases}$.

   We remark that these conditions can indeed be expressed a predicate ϕ on the α committed at Stage 2. For completeness, we show the formal definition of ϕ in Fig. 2. It is also worth noting that predicate ϕ needs to have the values r and s hard-wired, which are defined at Stages 3 and 4 respectively. This is why we need a Π_{ZKCnP} that allows us to defer the definition of the predicate until the **Prove** stage (Def. 8).
7. **(Receiver's Output).** R outputs $Y = (s_1, \ldots, s_n) \| (x_{b_1}, \ldots x_{b_t}) \| (b_1, \ldots, b_t)$.
8. **(Sender's Output).** S outputs $X = (x_1, \ldots, x_n) \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k}) \| (b_1, \ldots, b_t)$.

First, note that the $\mathsf{r} = \{b_1, \ldots, b_t\}$ sent by R in Stage 3 is a size-t random subset of $[n]$. It will overlap with $\{p_1, \ldots, p_k\}$ with negligible probability. Therefore, the **Editing** condition will almost never be triggered during a real execution of Prot. 1, thus can be safely ignored.

Stages 2 to 5 can be though as the following cut-and-choose procedure: the sender computes $\{y_i = f(x_i)\}_{i\in[n]}$; then the receiver checks t of them randomly. This ensures that a malicious S^* cannot cheat on more than $k = \delta n$ of the y_i's. We prove this statement formally in Claim 9, which requires us to handle extra technicalities due to the commit-and-prove structure and **Editing** condition. But this claim implies

Figure 2: Predicate $\phi_{\lambda,m,t,n,k,\mathsf{r},\mathsf{s}}(\cdot)$

Predicate ϕ has the values $(\lambda, m, t, n, k, \mathsf{r}, \mathsf{s})$ (as defined in Prot. 1) hard-wired. On the input α, $\phi_{\lambda,m,t,n,k,r,s}(\alpha) = 1$ if and only if all of the following hold:

- the α can be parsed as $M \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k})$, where $M = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}$ such that $|x_j| = \lambda$ and $|y_j| = m$ $\forall j \in [n]$, $|p_i| = \log(n)$ and $|y'_{p_i}| = m$ $\forall i \in [k]$; **and**
- the values $\{p_1, \ldots, p_k\}$ form a size-k subset of $[n]$; **and**
- the M_{r} consists of the columns in M specified by r; **and**
- the $\mathsf{s} = (s_1, \ldots, s_n)$ satisfy the following requirement (recall that the $\{b_1, \ldots, b_t\}$ are from r):
  - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then $s_i = y_i$ for all $i \in [n]$.
  - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then $s_i = \begin{cases} y'_i & i \in \{p_1, \ldots, p_k\} \\ y_i & i \in [n] \setminus \{p_1, \ldots, p_k\} \end{cases}$.

that a non-aborting Y output by an honest receiver contains at most $k = \delta n$ many s_i's that does *not* have a preimage under f (except with negligible probability). Let us assume w.l.o.g. that there are exactly k such "no-preimage" s_i's, which can be denoted as $\{s_{p_1}, \ldots, s_{p_k}\}$ (i.e. we denote the indices of these no-preimage s_i's by $\{p_1, \ldots, p_k\}$). Then, for each s_i where $i \in [n] \setminus \{p_1, \ldots, p_k\}$, this s_i must have (at least) one preimage under $f(\cdot)$. We denote an arbitrary preimage of such s_i as $f^{-1}(s_i)$. In particular, if i is equal to some $b_j \in \{b_1, \ldots, b_t\}$, the Y already contains the preimage for s_{b_j}, which is x_{b_j}.

We emphasize that, conditioned on $Y \neq \perp$, we have $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. To see this, recall that R checks at Stage 5 that $y_{b_i} = f(x_{b_i})$ *and* $s_{b_i} = y_{b_i}$ for all $b_i \in \{b_1, \ldots, b_t\}$. If there is a p_i falling in the set $\{b_1, \ldots, b_t\}$, then s_{p_i} $(= y_{p_i})$ does not have a preimage under $f(\cdot)$. Then, R will output $Y = \perp$ at Stage 5.

With these observations, we show in the following how to construct x and r such that $F^f(\mathsf{x} \| \mathsf{r}) = Y$. At a high-level, we take advantage of Case 2. We will use the no-preimage s_{p_i}'s together with their indices as the (p_i, y'_{p_i}) part in x. We will set r to the $\{b_1, \ldots, b_t\}$ contained in Y. Since $\{b_1, \ldots, b_t\} \cap \{p_1, \ldots, p_k\} = \emptyset$, the function F^f will put the no-preimage s_{p_i}'s at the positions specified by p_i's (according to Case 2), which will give us Y. Concretely, we set:

$$\mathsf{x} = (x'_1, \ldots, x'_n) \| (p_1, s_{p_1}), \ldots, (p_k, s_{p_k}) \text{ and } \mathsf{r} = (b_1, \ldots, b_t),$$

where x'_i's are defined as follows: $\forall i \in [n]$, $x'_i = \begin{cases} x_i & i \in \{b_1, \ldots, b_t\} \\ 0^\lambda & i \in \{p_1, \ldots, p_k\} \\ f^{-1}(s_i) & \text{otherwise} \end{cases}$.

We remark that $f^{-1}(s_i)$ may not be efficiently computable (indeed, f is a one-way function). But the above proof only relies on the *existence* of $f^{-1}(s_i)$. Also, we have $\{p_1 \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. It then follows from the description in Constr. 1 (in particular, Case 2) that $F^f(\mathsf{x} \| \mathsf{r}) = Y$.

The Full Proof. We now formally prove that the function F^f in Constr. 1 and Prot. 1 constitute a black-box OWF with proof (as per Def. 12). In Lem. 2, we prove that F^f satisfies the one-wayness requirement in Def. 12. Given Constr. 1, the completeness of Prot. 1 follows immediately by construction. We then establish the soundness and zero-knowledge property for Prot. 1 in Lem. 3 and 4 respectively.

Lemma 2 (One-Wayness). *The function F^f in Constr. 1 is one-way as defined in Def. 12.*

Proof. From the description of Constr. 1, it is easy to see that F^f is efficiently computable. In the following, we show that it is also "hard to invert".

Assume for contradiction that F^f is not hard to invert, i.e. there exist a PPT $\mathcal{A}_F$ and $\mathsf{r} \in \{0,1\}^{t\log(n)}$ such that for $\mathsf{x} \leftarrow \{0,1\}^{n\lambda+(\log(n)+m)k}$, $\mathcal{A}_F$ inverts $F^f(\mathsf{x}\|\mathsf{r})$ with non-negligible probability. We show how to construct a PPT $\mathcal{A}_f$ that inverts f with non-negligible probability.

On input y^*, $\mathcal{A}_f$ first samples a string $\mathsf{x} \leftarrow \{0,1\}^{n\lambda+(\log(n)+m)k}$ and computes the following Y value as per Constr. 1:

$$Y = F^f(\mathsf{x}\|\mathsf{r}) = (s_1, \ldots, s_n)\|(x_{b_1}, \ldots, x_{b_t})\|(b_1, \ldots, b_t).$$

It then samples $i^* \leftarrow [n] \setminus \{b_1, \ldots, b_t\}$, and gets Y' by substituting the s_{i^*} in Y with y^*. $\mathcal{A}_f$ then feeds Y' to $\mathcal{A}_F$ and receives X', which is supposed to be the preimage of Y' under F^f. In the following, we argue that X' contains the preimage of y^* under f with non-negligible probability.

Suppose that $\mathcal{A}_F$ produces an X' satisfying $F^f(X') = Y'$. This X' must be of the following form:

$$X' = (x_1, \ldots, x_n)\|(p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k})\|(b_1, \ldots, b_t).$$

Then, by Constr. 1, we must have $f(x_{i^*}) = y^*$ except for the "bad case" where $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$ *and* there is a $p_j = i^*$ (in which case x_{i^*} could be arbitrary as long as $\mathcal{A}_F$ sets $y'_{p_j} = y^*$). However, since i^* is picked uniformly from $[n] \setminus \{b_1, \ldots, b_t\}$, the "bad case" happens with probability $\leq 1/(n-t-k)$, which is $1/\mathsf{poly}(\lambda)$ by our choice of n, t, and k. Thus, if $\mathcal{A}_F$ breaks the one-wayness of F^f with some non-negligible probability $\varepsilon(\lambda)$, $\mathcal{A}_f$ will break the one-wayness of f with probability $\geq (1 - 1/\mathsf{poly}(\lambda)) \cdot \varepsilon(\lambda)$, which is non-negligible. □

Lemma 3 (Soundness.). *For every* PPT *machine* S^* *and every auxiliary input* $z \in \{0,1\}^*$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}\mathsf{r} \leftarrow \{0,1\}^{t\cdot\log(n)};\\ (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle\end{array} : \begin{array}{l}Y \neq \perp \textbf{ and}\\ \nexists \mathsf{x} \text{ s.t. } F^f(\mathsf{x}\|\mathsf{r}) = Y\end{array}\right] \leq \mathsf{negl}(\lambda). \tag{28}$$

Proof. Let us first define "non-trivial" S^*'s, which are the malicious provers that can make the honest receiver accepts with non-negligible probability.

Definition 13 (Non-Trivial S^*). *A PPT machine* S^* *is* non-trivial *if there exists some auxiliary input* $z \in \{0,1\}^*$ *such that the following holds: there exits a polynomial* $\mathsf{poly}(\cdot)$ *such that for infinitely many* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{r} \leftarrow \{0,1\}^{t\cdot\log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle : Y \neq \perp\right] \geq \frac{1}{\mathsf{poly}(\lambda)}. \tag{29}$$

It is not hard to see that if a PPT machine $S^*(1^\lambda, z)$ is *not* non-trivial, then Inequality (28) holds immediately. Therefore, to prove Lem. 3, we only need to focus on the non-trivial S^*'s.

For any Y of the form $(s_1, \ldots, s_n)\|(x_{b_1}, \ldots, x_{b_t})\|(b_1, \ldots, b_t)$, we define the following event:

- Bad_Y: there are more than $k = \delta n$ number of s_i's such that $\nexists x$ s.t. $f(x) = s_i$.

In the following, we present a claim (Claim 9). We will first show how to prove Lem. 3 assuming that Claim 9 holds and then present its proof.

Claim 9. *For every non-trivial* S^* *with the* $z \in \{0,1\}^*$ *satisfying Inequality (29), there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\mathsf{r} \leftarrow \{0,1\}^{t\cdot\log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle : \mathsf{Bad}_Y \;\middle|\; Y \neq \perp\right] \leq \mathsf{negl}(\lambda). \tag{30}$$

Consider the Y output by R from $\langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle$, where $S^*(1^\lambda, z)$ is non-trivial and $\mathsf{r} \leftarrow \{0,1\}^{t\log(n)}$. If $Y \neq \perp$, it must be of the following form:

$$Y = (s_1, \ldots, s_n)\|(x_{b_1}, \ldots, x_{b_t})\|(b_1, \ldots, b_t),$$

where $s_i = f(x_i)$ for all $i \in \{b_1, \ldots, b_t\}$. Assuming that Claim 9 holds, to finish the proof of Lem. 3, it suffices to show the following claim:

– Conditioned on $Y \neq \bot$, if Bad_Y does not happen, then there exist an x and an r such that $F^f(\mathsf{x}\|\mathsf{r}) = Y$.

Conditioned on $Y \neq \bot$, Claim 9 implies that there are at most $k = \delta n$ many s_i's that do not have a preimage under f (except with negligible probability). In the following, we assume w.l.o.g. that there are exactly k such "no-preimage" s_i's. We denote them as $\{s_{p_1}, \ldots, s_{p_k}\}$ (i.e. we denote the indices of these no-preimage s_i's by $\{p_1, \ldots, p_k\}$). Then, for each s_i where $i \in [n] \setminus \{p_1, \ldots, p_k\}$, this s_i must have (at least) one preimage under $f(\cdot)$. We denote an arbitrary preimage of such s_i as $f^{-1}(s_i)$. In particular, if i is equal to some $b_j \in \{b_1, \ldots, b_t\}$, then Y already contains the preimage for s_{b_j}, which is x_{b_j}.

We emphasize that, conditioned on $Y \neq \bot$, $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. To see this, recall that R checks at Stage 5 that $y_{b_i} = f(x_{b_i})$ *and* $s_{b_i} = y_{b_i}$ for all $b_i \in \{b_1, \ldots, b_t\}$. If there is a p_i falling in the set $\{b_1, \ldots, b_t\}$, then s_{p_i} $(= y_{p_i})$ does not have a preimage under $f(\cdot)$. Then, R will output $Y = \bot$ at Stage 5.

With these observations, we show in the following how to construct x and r such that $F^f(\mathsf{x}\|\mathsf{r}) = Y$, assuming that Bad_Y does not happen. At a high-level, we take advantage of Case 2. we will use the no-preimage s_{p_i}'s together with their indices as the (p_i, y'_{p_i}) part in x. We will set r to the $\{b_1, \ldots, b_t\}$ contained in Y. Since $\{b_1, \ldots, b_t\} \cap \{p_1, \ldots, p_k\} = \emptyset$, the function F^f will put the no-preimage s_{p_i}'s at the positions specified by p_i's (according to Case 2), which will give us Y. Concretely, we set:

$$\mathsf{x} = (x'_1, \ldots, x'_n)\|(p_1, s_{p_1}), \ldots, (p_k, s_{p_k}) \text{ and } \mathsf{r} = (b_1, \ldots, b_t),$$

where x'_i's are defined as follows: $\forall i \in [n]$, $x'_i = \begin{cases} x_i & i \in \{b_1, \ldots, b_t\} \\ 0^\lambda & i \in \{p_1, \ldots, p_k\} \\ f^{-1}(s_i) & \text{otherwise} \end{cases}$. We remark that $f^{-1}(s_i)$ may not be efficiently computable (indeed, f is a one-way function). But this proof only relies on the *existence* of $f^{-1}(s_i)$. Also, we have $\{p_1 \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. It then follows from the description in Constr. 1 (in particular, Case 2) that $F^f(\mathsf{x}\|\mathsf{r}) = Y$.

In the following, we show the proof for Claim 9, which will complete the proof for Lem. 3.

Proof of Claim 9. All the probabilities appearing in this proof are taken over the following random procedure:

$$\mathsf{r} \leftarrow \{0,1\}^{t \cdot \log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r}) \rangle,$$

where S^* and z are as described in Claim 9.

First, note that $\Pr\left[\mathsf{Bad}_Y \,\middle|\, Y \neq \bot\right] \cdot \Pr[Y \neq \bot] = \Pr[\mathsf{Bad}_Y \wedge (Y \neq \bot)]$. Since S^* is non-trivial, we know from Inequality (29) that $\Pr[Y \neq \bot]$ is non-negligible. Therefore, to prove Inequality (30), it suffices to show

$$\Pr[\mathsf{Bad}_Y \wedge (Y \neq \bot)] \leq \mathsf{negl}(\lambda), \tag{31}$$

which we prove in the following.

Consider the execution $(\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r}) \rangle$ where $Y \neq \bot$. Observe that S^* at Stage 2 commits a value α[11] of the form shown in Expression (27). For this execution, we define the following sequence of events:

– E_1: $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. The $\{p_i\}_{i \in [k]}$ are contained in α and $\{b_i\}_{i \in [t]}$ are contained in Y.

– E_2: $\forall i \in [n]$, $y_i = s_i$. The $\{y_i\}_{i \in [n]}$ are contained in the second row of M (which is in turn contained in α), and $\{s_i\}_{i \in [n]}$ are contained in Y.

– E_3: The M_{r} sent by S^* at Stage 4 indeed consists of the columns of M specified by $\mathsf{r} = \{b_1, \ldots, b_t\}$ (contained in Y).

– E_4: there are more than $k = \delta n$ number of y_i's (contained in the second row of M) that do not have a preimage under $f(\cdot)$.

[11] This α is well-defined as BBCom is statistically-binding.

We first claim:

$$\Pr[E_4 \wedge (Y \neq \bot)] \leq \mathsf{negl}(\lambda). \tag{32}$$

To see that, first notice the following:

$$\begin{aligned}\Pr[E_4 \wedge (Y \neq \bot)] &= \Pr[E_4 \wedge (Y \neq \bot) \wedge E_3] + \Pr\left[E_4 \wedge (Y \neq \bot) \wedge \overline{E}_3\right] \\ &\leq \underbrace{\Pr\left[(Y \neq \bot) \;\middle|\; E_3 \wedge E_4\right]}_{\mathsf{P}_1} + \underbrace{\Pr\left[(Y \neq \bot) \wedge \overline{E}_3\right]}_{\mathsf{P}_2}.\end{aligned}$$

We now show that both P_1 and P_2 are negligible. First, recall that R checks at Stage 5 if $y_i = f(x_i)$ for all columns $[x_i \; y_i]^T$ contained in M_{r}. Thus, conditioned on E_3, the receiver does not abort only if the set $\mathsf{r} = \{b_1, \ldots, b_t\}$ *does not* select any "bad" column $[x_i \; y_i]^T$ in M s.t. $y_i \neq f(x_i)$. Moreover, E_4 ensures that there are more than $k = \delta n$ such "bad" columns in M. Therefore, the probability P_1 is smaller than $(1-\delta)^t$, which is negligible as $0 < \delta < 1$ is a constant and t is $\omega(\log \lambda)$.

Also, observe that E_3 was actually proved at Stage 6 (as Item 6b) by S^* via the commit-and-prove protocol Π_{ZKCnP}. The soundness of Π_{ZKCnP} implies that P_2 is negligible.

Next, we make another claim:

$$\Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \leq \mathsf{negl}(\lambda). \tag{33}$$

To prove this inequality, notice the following:

$$\begin{aligned}\Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] &= \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2 \wedge E_1\right] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2 \wedge \overline{E}_1\right] \\ &\leq \underbrace{\Pr[E_1]}_{\mathsf{P}_3} + \underbrace{\Pr\left[(Y \neq \bot) \wedge \overline{E}_2 \;\middle|\; \overline{E}_1\right]}_{\mathsf{P}_4}.\end{aligned}$$

We now show that both P_3 and P_4 are negligible. Recall that E_1 is the event that the set $\{p_1, \ldots, p_k\}$ contained in α does not overlap with $\mathsf{r} = \{b_1, \ldots, b_t\}$. First, note that the Π_{ZKCnP} proof at Stage 6 ensures that the set $\{p_1, \ldots, p_k\}$ is a size-k subset of $[n]$ (i.e. Item 6a) except with negligible probability. Also, observe that the r is a size-t random subset of $[n]$ that is sampled *independently* of $\{p_1, \ldots, p_k\}$. Therefore, E_1 happens with probability $\leq (1-\delta)^t + \mathsf{negl}(\lambda)$, which is negligible.

According to Constr. 1, if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$ (which is exactly the event $\overline{E}_1$), then it must hold that $y_i = s_i$ for all $i \in [n]$ (which is exactly E_2). Also, recall that this condition is enforced by the $\mathsf{BBProve}$ performed by S^* at Stage 6 (see Item 6c). The soundness of the Π_{ZKCnP} guarantees that, conditioned on $\overline{E}_1$, if E_2 does not hold, then R will abort with overwhelming probability. Therefore, P_4 is negligible.

We are now ready to derive Inequality (31):

$$\begin{aligned}\Pr[\mathsf{Bad}_Y \wedge (Y \neq \bot)] &= \Pr[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge E_2] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \\ &= \Pr[E_4 \wedge (Y \neq \bot) \wedge E_2] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \qquad (34) \\ &\leq \Pr[E_4 \wedge (Y \neq \bot)] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \\ &\leq \mathsf{negl}(\lambda) \qquad (35)\end{aligned}$$

where Eq. (34) follows from the fact that, conditioned on E_2, the events Bad_Y and E_4 are identical. Also, note that Inequality (35) follows from Inequalities (32) and (33).

This finishes the proof of Claim 9 and thus also the proof for Lem. 3. □

Lemma 4 (Zero-knowledge). *Prot. 1 satisfies the zero-knowledge property as in Def. 12.*

Proof. To prove the zero-knowledge property, we need to show a PPT ideal-world simulator Sim for any PPT malicious receiver R^*. At a high-level, such a simulator can be constructed as follows. Sim will use the simulator of the commit-and-prove protocol Π_{ZKCnP} at Stages 2 and 6. This allows Sim to finish the interaction without knowing the sender's input x.

Formally, we will show a sequence of hybrids starting from the real execution between the honest sender and R^*, and show that the last hybrid is essentially the simulator we want. We use Out_{H_i} to denote the output of hybrid H_i.

Hybrid $H_0(1^\lambda, \mathsf{x}, z)$**.** This hybrid uses the strategy of the honest $S(1^\lambda, \mathsf{x})$ to interact with the corrupted receiver $R^*(1^\lambda, z)$. At the end of the execution, H_0 outputs whatever R^* outputs. This hybrid is exactly the real execution.

Hybrid $H_1(1^\lambda, \mathsf{x}, z)$**.** This hybrid is identical to the previous one, except that

- At Stage 2, instead of executing BBCom, H_1 uses the strategy of Sim_1 in its communication with R^*, where Sim_1 is the simulator for the **Commit** stage of Π_{ZKCnP} (see Def. 8).
- At Stage 6, instead of doing the proof honestly, H_1 uses the strategy of Sim_2in its communication with R^*, where Sim_1 is the simulator for the **Prove** stage of Π_{ZKCnP} (see Def. 8).

$\mathsf{Out}_{H_0} \overset{c}{\approx} \mathsf{Out}_{H_1}$: This is due to the ZK property of Π_{ZKCnP}.

The simulator $\mathsf{Sim}(1^\lambda, z)$**.** We now describe the simulator Sim for the ideal execution. Sim is identical to H_1, except that

- Sim does not need to execute Stage 1;
- Upon receiving r at Stage 3, Sim sends it to the idea functionality $\mathcal{F}_{F^f}$, and receives back the value $Y = (s_1, \ldots, s_n) \| (x_{b_1}, \ldots, x_{b_t}) \| (b_1, \ldots, b_t)$;
- At Stage 4, Sim sets $\widetilde{M}_\mathsf{r} = \begin{bmatrix} x_{b_1} & \cdots & x_{b_t} \\ s_{b_1} & \cdots & s_{b_t} \end{bmatrix}$, and sends to R the values $\widetilde{M}_\mathsf{r}$ and $\mathsf{s} = (s_1, \ldots, s_n)$, where the s_i's are those contained in Y.

We remark that, unlike H_1, Sim does not need to know the input x to the sender; the Y that it obtained from $\mathcal{F}_{F^f}$ contains all the necessary information to finish its interaction with R^*. In particular, although the x_{b_i}'s in $\widetilde{M}_\mathsf{r}$ and the s now come from the Y that Sim obtained from the ideal functionality, they are identically distributed to the ones generated by the honest sender in a real execution. It then follows that the output of Sim is identical to H_1.

This finishes the proof for Lem. 4. □

6 Proof-Based Pseudo-Random Generators (Overview)

We can also define proof-based pseudo-random generators (PB-PRGs) in a similar way as for PB-OWFs. It consists of a two-input function $G^g(\cdot, \cdot)$ and a protocol $\Pi^g_G = (S^g, R^g)$ such that for any PRG g, $G^g(\cdot, \mathsf{r})$ is a PRG for any choice of r, and Π^g_G satisfies the same completeness, soundness, and ZK requirements as in Def. 12 but w.r.t. G^g.

Our PB-PRG can be constructed by simply replacing the oracle OWF f with a PRG g in both Constr. 1 and Prot. 1 (our PB-OWF construction). There is one caveat: the output Y of Constr. 1 contains the preimage x_{b_i} for y_{b_i} (or s_{b_i}). While this is fine for one-wayness, such a Y will not be pseudo-random, because an adversary can always learn if Y is in the range of $G^g(\cdot, \mathsf{r})$ by testing whether $y_{b_i} = g(x_{b_i})$. To fix this, in the output Y, we will place x_{b_i} in the position where we originally put y_{b_i} (and we can drop the $(x_{b_1}, \ldots, x_{b_t})$ part from Y). We will show that this modification lead to a valid PB-PRG.

We present the definition, construction, and the security proof for PB-PRGs in Appx. A, due to their pronounced similarity to our PB-OWF.

7 Proof-Based Collision-Resistant Hash Families

We now discuss proof-based collision-resistant hash families (PB-CRHFs). As mentioned in Sec. 2.3, the definition and construction of PB-CRHF follow the same template as our PB-OWFs, except that we need to handle the **Editing** condition differently.

7.1 Definition

Our PB-CRHF consists of an oracle machine $H^{(\cdot)}$ and an oracle protocol $\Pi_H^{(\cdot)}$. As mentioned in Sec. 2.3, the $H^{(\cdot)}$ will be instantiated as a hash *family*. That is, given a collision-resistant hash family $\mathcal{H}'$, we first run its KGen' to sample a function $h_i \in \mathcal{H}'$, and then instantiate $H^{(\cdot)}$'s oracle as h_i. Therefore, H^{h_i} is also a hash family whose KGen simply runs KGen' for $\mathcal{H}'$ (and samples a random string z that we will describe later).

Once $H^{(\cdot)}$ and $\Pi_H^{(\cdot)}$ are instantiated with an $h_i \leftarrow \mathsf{KGen}'(1^\lambda)$, we can start talking about security. Same as in Def. 12, H^{h_i} takes two inputs x and r. We require that, for all r, $H^{h_i}(\cdot, \mathsf{r})$ is collision-resistant on its first input. The protocol $\Pi_H^{h_i}$ satisfies similar completeness, soundness, and ZK requirements as those in Def. 12. We provide the formal definition in Def. 14.

Definition 14 (Proof-Based CRHF). *Let $a(\lambda)$, $b(\lambda)$ and $c(\lambda)$ be polynomials on λ such that $a(\lambda)+b(\lambda) < c(\lambda)$. A proof-based collision-resistant hash family (PB-CRHF) is a function family $\mathcal{H} = \{H_i\}_{i\in I}$ for some index set I, where $H_i : \{0,1\}^{a(\lambda)} \times \{0,1\}^{b(\lambda)} \to \{0,1\}^{c(\lambda)}$. For each $H_i \in \mathcal{H}$, there exists a protocol $\Pi_i = (S, R)_i$ consisting of a pair of* PPT *machines. $\mathcal{H}$ is a collision-resistant hash family in the following sense.:*

- **Collision-Resistance.** *$\mathcal{H}$ has PPT algorithms KGen and Eval, which satisfy the same requirements as in Def. 1; it further satisfies the following collision-resistant requirement: for any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$,*

$$\Pr\left[i \leftarrow \mathsf{KGen}(1^\lambda), (\mathsf{x}, \mathsf{x}') \leftarrow \mathcal{A}(1^\lambda, i) : \mathsf{x} \neq \mathsf{x}' \wedge H_i(\mathsf{x}\|\mathsf{r}) = H_i(\mathsf{x}'\|\mathsf{r})\right] \leq \mathsf{negl}(\lambda).$$

Let $(X, Y) \leftarrow \langle S(1^\lambda, \mathsf{x}), R(1^\lambda, \mathsf{r})\rangle_i$ denote the execution of the protocol Π_i, where the security parameter is λ, the input to S and R are x and r respectively, and the output of S and R are X and Y respectively. Let $Y = \perp$ denote the event that R aborts in the execution. The following properties for protocol $\Pi_i = (S, R)_i$ hold with overwhelming probability over the choice of $i \leftarrow \mathsf{KGen}(1^\lambda)$ (see also Rmk. 5):

- **Completeness.** *$\forall \lambda \in \mathbb{N}$, $\forall i \in \{0,1\}^\lambda$, $\forall \mathsf{x} \in \{0,1\}^{a(\lambda)}$ and $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$,*

$$\Pr\left[(X, Y) \leftarrow \langle S(1^\lambda, \mathsf{x}), R(1^\lambda, \mathsf{r})\rangle_i : X = \mathsf{x}\|\mathsf{r} \textbf{ and } Y = H_i(\mathsf{x}\|\mathsf{r})\right] = 1,$$

  where the probability is taken over the the randomness used by S and R.

- **Soundness.** *For every* PPT *machine S^* and every auxiliary input $\mathsf{aux} \in \{0,1\}^*$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\begin{array}{l|l} \mathsf{r} \leftarrow \{0,1\}^{b(\lambda)}; & Y \neq \perp \textbf{ and} \\ (\cdot, Y) \leftarrow \langle S^*(1^\lambda, \mathsf{aux}), R(1^\lambda, \mathsf{r})\rangle_i & \nexists \mathsf{x} \text{ s.t. } H_i(\mathsf{x}\|\mathsf{r}) = Y \end{array}\right] \leq \mathsf{negl}(\lambda)$$

  where the probability is taken over the random sampling of i and r,and the randomness used by S^* and R.

- **Zero-Knowledge.** *This property is defined as* only *requiring security against corrupted R in the ideal-real paradigm for 2PC w.r.t. the ideal functionality $\mathcal{F}_{H_i}$ in Fig. 3. Namely, there exist a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$, $\forall \mathsf{aux} \in \{0,1\}^*$, $\forall \mathsf{x} \in \{0,1\}^{a(\lambda)}$ and $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$,*

$$\mathsf{REAL}_{\Pi_i, \mathcal{A}(\mathsf{aux})}(\mathsf{x}, \mathsf{r}) \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}_{H_i}, \mathsf{Sim}(\mathsf{aux})}(\mathsf{x}, \mathsf{r}),$$

  where $\mathsf{REAL}_{\Pi_i, \mathcal{A}(\mathsf{aux})}(\mathsf{x}, \mathsf{r})$ and $\mathsf{IDEAL}_{\mathcal{F}_{H_i}, \mathsf{Sim}(\mathsf{aux})}(\mathsf{x}, \mathsf{r})$ are defined in the same way as in Def. 12.

Figure 3: Functionality $\mathcal{F}_{H_i}$ for Proof-Based CRHFs

The ideal functionality $\mathcal{F}_{H_i}$ interacts with a sender S and a receiver R. Upon receiving the input $\mathsf{x} \in \{0,1\}^{a(\lambda)}$ from S and $\mathsf{r} \in \{0,1\}^{b(\lambda)}$ from R, the functionality $\mathcal{F}_{H_i}$ sends $\mathsf{x}\|\mathsf{r}$ to S, and $H_i(\mathsf{x}\|\mathsf{r})$ to R.

If both the constructions of $\mathcal{H}$ and Π only involve black-box access to other primitives, we obtain black-box *PB-CRHF.*

Remark 5. Similar to the case of PB-OWFs, the protocol here is meant to be a ZK system for the range-membership of the function. But note that $\mathcal{H}$ in the above PB-CRHF is a *family* of functions, and the concrete function H_i is sampled by running the key generation algorithm. So, the range-membership is well-defined only when H_i is sampled and fixed. Therefore, the security properties of protocol Π_i will depend on the sampling of H_i by running KGen. This idea already appears explicitly in the definition of collision-resistant hash *families*—the adversary's winning probability in the collision-resistant game depends on the random procedure of executing KGen.

7.2 Merkle Tree Related Notation

As mentioned in the technical overview (Sec. 2.3), our construction makes use of the Merkle hashing tree [Mer90]. In this part, we setup related notation.

Notation for Perfect Binary Trees. A *perfect* binary tree is a binary tree in which all interior nodes have two children and all leaves have the same depth. A perfect binary tree of hight ℓ has $2^{\ell+1}-1$ nodes, where 2^ℓ of the nodes are leaves. There exist canonical methods (e.g., array representation) to index the nodes in the tree, which forms a bijection between the nodes and the set $[2^{\ell+1}-1]$. We will use the indices to refer to the corresponding nodes under this bijection. In particular, we require that the first 2^ℓ indices (i.e. $[2^\ell]$) represent the leaves.

For each node $k \in [2^{\ell+1}-1]$, v_k denotes the content/value of this node. For a leaf node $i \in 2^\ell$, the *sibling path* of i consists of the value v_i together with the values of all the siblings of nodes on the path from i to the root. We use P_i to denote the sibling path of leaf i, and $\mathsf{Ind}(\mathsf{P}_i)$ denotes the collection of indices of the nodes on path P_i. For k sibling paths $(\mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_k})$, we use $\mathsf{Ind}(i_1, \ldots, i_k)$ to denote the collection of indices of the nodes on all these paths, i.e.,

$$\mathsf{Ind}(i_1, \ldots, i_k) \coloneqq \mathsf{Ind}(\mathsf{P}_{i_1}) \cup \ldots \cup \mathsf{Ind}(\mathsf{P}_{i_1}). \tag{36}$$

We remark that $\mathsf{Ind}(i_1, \ldots, i_k)$ is a set of indices for nodes. It does not depend on the contents/values of nodes. Once the structure of the tree is fixed, this set is then fixed, even if the tree contains only dummy nodes.

Merkle Trees. A Merkle hashing tree [Mer90] is a special type of perfect binary tree constructed as follows. Let $n = 2^l$ for some integer l. Given a hash function $h : \{0,1\}^{2m} \to \{0,1\}^m$ and a length-mn string X, the Merkle tree $MT_{h,m}(X)$ is a size-$(2n-1)$ perfect binary tree of the following form:

- X is parsed as n blocks $(x_1, \ldots, x_n)$, each x_i is of length m. These x_i's are the contents of the n leaves;
- **(Merkle Consistency.)** For any non-leaf node k and its left child ℓ and right child r, their contents satisfy the equation $h(v_k) = h(v_\ell \| v_r)$.

A sibling path P_i ($i \in [n]$) is said to be *Merkle-consistent* if all the nodes on this path satisfy the above Merkle consistency condition.

7.3 Our Construction

Construction 2: Collision-Resistant Hash Family $H_z^{h_i}$

Let $m(\lambda)$ and $n(\lambda)$ be polynomials of λ. Assume w.l.o.g. that is n a power of 2 (i.e., $n = 2^\ell$ for some ℓ). Let $0 < \delta < 1$ be a constant, and $k(\lambda) = \delta n(\lambda)$. Let $t(\lambda) = \log^2(\lambda)$ (see also Rmk. 4). Let $\mathcal{H}' = \{h_i\}_{i\in I}$ be a collision-resistant hash family where $h_i : \{0,1\}^{2m(\lambda)} \to \{0,1\}^{m(\lambda)}$. Denote its key generation as KGen'.

- **Key Generation.** On input 1^λ, sample a function from $\mathcal{H}'$ by running $h_i \leftarrow \mathsf{KGen}'(1^\lambda)$; sample a random string $z \leftarrow \{0,1\}^{m(\lambda)}$; outputs (i, z) as the hash key.
- **Evaluation.** On input $\mathsf{x} \in \{0,1\}^{nm+k(\log(2n)+m)+3m}$ and $\mathsf{r} \in \{0,1\}^{t\log(n)}$, the evaluation algorithm parses them as:

$$\mathsf{x} = (x_1, \ldots, x_n) \| (p_1, v_{p_1}), \ldots, (p_k, v_{p_k}) \| \tau \| \mu, \quad \mathsf{r} = (b_1, \ldots, b_t), \tag{37}$$

  where $|x_i| = |v_{p_i}| = m$, $\{p_i\}_{i\in[k]}$ is a size-k subset of $[2n-1]$, and $\{b_i\}_{i\in[t]}$ is a size-t subset of $[n]$. The set $\{b_i\}_{i\in[t]}$ specifies t leaves out of all the n leaves.
  The algorithm builds a perfect binary tree T that has n leaves, where all the nodes are dummies. As discussed in Sec. 7.2, the indices of the nodes in T are well-defined, even though T now contains only dummy nodes. The evaluation procedure outputs $Y = \mathsf{t}_1 \| \mathsf{t}_2 \| (\mathsf{P}_{b_1}, \ldots, \mathsf{P}_{b_t}) \| (b_1, \ldots, b_t)$, which is computed as follows:

  1. **Non-Editing:** If $\tau = z$ **or** $h_i(\tau) \neq h_i(z)$ **or** $\mathsf{Ind}(b_1, \ldots, b_t) \cap \{p_1, \ldots, p_k\} \neq \emptyset$:
     (a) It fills the tree T as follows. It places $(x_1, \ldots, x_n)$ at the n leaves. For any other node in T, its content is the hash value under h_i on the concatenation of its left child and right child. Denote the root value as t_1.
     (b) For $i \in [t]$, P_{b_i} is the sibling path of leaf x_{b_i} in the above tree T;
     (c) Use h_i to hash[a] the following Λ value to a length-m string denoted as t_2:

$$\Lambda = (p_1, v_{p_1}), \ldots, (p_k, v_{p_k}) \| \tau \| \mu.$$

  2. **Editing:** if $\tau \neq z$ **and** $h_i(\tau) = h_i(z)$ **and** $\mathsf{Ind}(b_1, \ldots, b_t) \cap \{p_1, \ldots, p_k\} = \emptyset$:
     (a) It fills the tree T as follows. It places $(x_1, \ldots, x_n)$ at the n leaf positions in T. Then, fill the tree bottom up, following the rule for Merkle tree (i.e. the hashing of two children nodes' contents is the parent node's content), with the following exception: for node $p_i \in \{p_1, \ldots, p_k\}$, it fills node p_i with the v_{p_i} contained in x instead of the hash of the children of node p_i. Denote the root value as t_1.
     (b) For $i \in [t]$, P_{b_i} is defined as the sibling path of leaf x_{b_i} in the tree T;
     (c) Set $\mathsf{t}_2 = \mu$ (recall that μ is contained in x);

[a] Note that the input to h_i should have length $2m$. But $|\Lambda| > 2m$. This can be handled using domain-extension techniques, e.g., the Merkle-Damgård transformation [Mer90, Dam90].

The formal construction is provided in Constr. 2 and Prot. 2. We follow the high-level idea described in Sec. 2.3 with the following modifications. Instead of hashing the $(x_1, \ldots, x_n)$ (contained in x) separately, we build a Merkle tree using them as the leaves. In Constr. 2, P_i denotes the sibling path from leaf x_i to the root; $\mathsf{Ind}(b_1, \ldots, b_t)$ denotes the set of *indices* of the nodes on path $\mathsf{P}_{b_1}, \ldots, \mathsf{P}_{b_t}$. (See Sec. 7.2 for relevant notation.) In Prot. 2, the receiver checks t leaves and their corresponding sibling paths. This ensures that there are at least $(n-k)$ "good" leaves, in the sense that there are valid sibling paths from the Merkle root to them. In the **Editing** case, this will allow us to perform preimage editing by planting the v_{p_i} values on the k "bad" paths to obtain a (partial) tree consistent with the root t_1 contained in Y. Note that we also hash the Λ in Step 1c. As explained in Sec. 2.3, this is to prevent the adversary from taking advantage of preimage editing to find collisions.

Protocol 2: Protocol $\Pi_z^{h_i}$ for Our PB-CHRF

Let $\mathcal{H}'$, m, n, δ, t and k be as in Constr. 2. Let $\Pi_{\mathsf{ZKCnP}} = (\mathsf{BBCom}, \mathsf{BBProve})$ be a black-box commit-and-prove protocol. For a function defined by (i, z) from the PB-CRHF in Constr. 2, this protocol proceeds as follows. Both parties take the security parameter 1^λ as the common input. Sender S takes a string $\mathsf{x} \in \{0,1\}^{nm+k(\log(n)+m)+3m}$ as private input; receiver R takes a string $\mathsf{r} \in \{0,1\}^{t\log(n)}$ as private input.

1. S parses x as $(x_1, \ldots, x_n) \| (p_1, v_{p_1}), \ldots, (p_k, v_{p_k}) \| \tau \| \mu$ (in the same manner as in Expression (37)). S build a Merkle tree $MT'_{h,m}(x)$ using $(x_1, \ldots, x_n)$ as the leaves (this is identical to Step 1a). Denote the root of this tree as t_x.
2. S and R execute $\mathsf{BBCom}(\nu)$, the **Commit** stage of Π_{ZKCnP}, where S commits to the following value
$$\nu \coloneqq \mathsf{t}_\mathsf{x} \| (p_1, \ldots, p_k). \tag{38}$$
3. R sends the value r.
4. S parses r as $(b_1, \ldots, b_t)$ where each b_i is of length $\log(n)$. With the values x and r, S evaluates the function $H_z^{h_i}$ as per Constr. 2 to compute the following Y, which it sends to R:
$$Y = \mathsf{t}_1 \| \mathsf{t}_2 \| (\mathsf{P}_{b_1}, \ldots, \mathsf{P}_{b_t}) \| (b_1, \ldots, b_t).$$
5. R checks if P_{b_i} is Merkle-consistent for all $i \in [t]$. R aborts if any of the check fails.
6. S and R execute $\mathsf{BBProve}$, the **Prove** stage of Π_{ZKCnP}, where S proves that the ν committed in Stage 2 satisfies the following conditions:
   (a) the $\{p_1, \ldots, p_k\}$ in ν form a size-k subset of $[2n-1]$, where $k = \delta n$; **and**
   (b) the t_x contained in τ is equal to t_1, **or** $\mathsf{Ind}(b_1, \ldots, b_t) \cap \{p_1, \ldots, p_k\} = \emptyset$.
7. **(Receiver's Output).** R outputs $Y = \mathsf{t}_1 \| \mathsf{t}_2 \| (\mathsf{P}_{b_1}, \ldots, \mathsf{P}_{b_t}) \| (b_1, \ldots, b_t)$.
8. **(Sender's Output).** S outputs $X = (x_1, \ldots, x_n) \| (p_1, v_{p_1}), \ldots, (p_k, v_{p_k}) \| \tau \| \mu \| (b_1, \ldots, b_t)$.

It is also worth noting that Constr. 2 and Prot. 2 work for an x of fixed length. But since we hash the $\{x_i\}_{i\in[n]}$ part using a Merkle tree, we can handle x with a various-length $\{x_i\}_{i\in[n]}$ part (which dominates the length of x). To maintain security, we simply include the height of the Merkle tree in Y.

7.4 Proof of Security

In this section, we prove the following theorem.

Theorem 5. *The construction shown in Constr. 2 and Prot. 2 is a PB-CRHF (as per Def. 14) that makes only black-box use of a CRHF $\mathcal{H}'$. Moreover, the PB-OWF is private-coin (resp. public-coin) if $\mathcal{H}'$ is private-coin (resp. public-coin).*

It follows immediately from the description that our construction makes only black-box use of $\mathcal{H}'$, and it also satisfies the complement requirement as per Def. 14. In the following, we first prove the collision resistance of Constr. 2 in Lem. 5. Then, we establish the soundness and zero-knowledge property for Prot. 2 in Lem. 6 and Lem. 7, respectively.

Lemma 5 (Collision Resistance). *Constr. 2 satisfies the collision-resistant requirement as in Def. 14. Moreover, Constr. 2 is private-coin (resp. public-coin) if $\mathcal{H}'$ is private-coin (resp. public-coin).*

Proof. We first show that the function is input-compressing. According to Constr. 2. the input is of length:
$$|X| = |\mathsf{x}| + |\mathsf{r}| = nm + k(\log(2n) + m) + 3m + t\log(n),$$

and the output is the length

$$|Y| = 2m + 2\ell m + t\log(n),$$

where $\ell = \log(n)$ is the height of the Merkle tree T. Thus, we have $|X| > |Y|$.

Public-Coin v.s. Private-Coin. Let us recall the key generation algorithm in Constr. 2. In addition to running the KGen' of $\mathcal{H}'$, it only sample a random string $z \in \{0,1\}^{m(\lambda)}$, which is a public-key operation. Therefore, this key generation procedure is private-coin (resp. public-coin) if $\mathcal{H}'$ is private-coin (resp. public-coin).

Collision Resistance. Assume for contradiction that there is a PPT adversary $\mathcal{A}$ that breaks the collision-resistance property of Constr. 2.

Consider a function $H_{h_i,z}$ sampled from the family, where $h_i \leftarrow \mathsf{KGen}(1^\lambda)$ and $z \leftarrow \{0,1\}^{2m}$. First, we claim that $\mathcal{A}$ cannot output an X that contains a τ satisfying $\tau \neq z$ and $h_i(\tau) = h_i(z)$; otherwise, $\mathcal{A}$ can be used to break the collision resistance of h. Therefore, we assume in the following that the purported collision pair X, X' output by $\mathcal{A}$ will not trigger the **Editing** condition.

Conditioned on the event that the **Editing** condition is not triggered, if $H_{h_i,z}(X) = H_{h_i,z}(X')$, X and X' must differ in the $(x_1, \ldots, x_n)$ part. This is because the output of $H_{h_i,z}$ contains the values $(b_1, \ldots, b_t)$ (thus, the collision will not happen here); it also contains a hash of the value $\Lambda = (p_1, v_{p_1}), \ldots, (p_k, v_{p_k})\|\tau\|\mu$. If X and X' have different Λ's, $\mathcal{A}$ breaks the collision-resistant property of h_i.

The above argument implies that X and X' contain different $(x_1, \ldots, x_n)$ values, but $H^{h_i}(X)$ and $H^{h_i}(X')$ contain the same t_1, the root of the Merkle tree on $(x_1, \ldots, x_n)$. Therefore, $\mathcal{A}$ can be converted to a collision-finder that breaks the collision-resistant of h_i (in the Step 1a Merkle tree). □

Completeness follows immediately from the description of Constr. 2 and Prot. 2. In the following, we prove soundness (Lem. 6) and zero-knowledge (Lem. 7).

Lemma 6 (Soundness). *Prot. 2 satisfies the soundness requirement defined in Def. 14.*

Proof. First, let $\mathsf{Bad}_{\mathsf{t}_1}$ denote the following event: there does not exist an n-leaf full and complete binary tree rooted at t_1 (contained in Y) such that at least $(1-\delta)n$ leaves with their corresponding sibling paths satisfy the Merkle consistency requirement.

At a high-level, the proof for Lem. 6 follows the approach of that for Lem. 3. First, we assert in Claim 10 that it is impossible (except with negligible probability) that R accepts the execution *and* the event $\mathsf{Bad}_{\mathsf{t}_1}$ happens. Second, we show that whenever R accepts and $\mathsf{Bad}_{\mathsf{t}_1}$ does not happen, there must exist a valid preimage for the Y learned by R. These two claims together imply Lem. 6.

Claim 10. *For every* PPT *machine S^* and every auxiliary input $\mathsf{aux} \in \{0,1\}^*$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\mathsf{r} \leftarrow \{0,1\}^{t\log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, \mathsf{aux}), R(1^\lambda, \mathsf{r})\rangle : (Y \neq \perp) \wedge \mathsf{Bad}_{\mathsf{t}_1}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random sampling of r,and the randomness used by S^* and R.

Proof. We claim that, except with negligible probability, the t_1 in Y that S^* sends in Stage 4 is equal to the t_x in ν committed in Stage 2. Due to the soundness of the $\mathsf{BBProve}$ performed in Stage 6, the following holds with overwhelming probability

$$\mathsf{t}_1 = \mathsf{t}_\mathsf{x}, \quad \textbf{or} \quad \mathsf{Ind}(b_1, \ldots, b_t) \cap \{p_1, \ldots, p_k\} = \emptyset.$$

Moreover, since the values $\{b_i\}_{i=1}^t$ form a *random* size-t subset of $[n]$, the event $\mathsf{Ind}(b_1, \ldots, b_t) \cap \{p_1, \ldots, p_k\} = \emptyset$ will only happen with probability $\leq (1-\delta)^t$ (recall that $k = \delta n$), which is negligible as $0 < \delta < 1$ is a

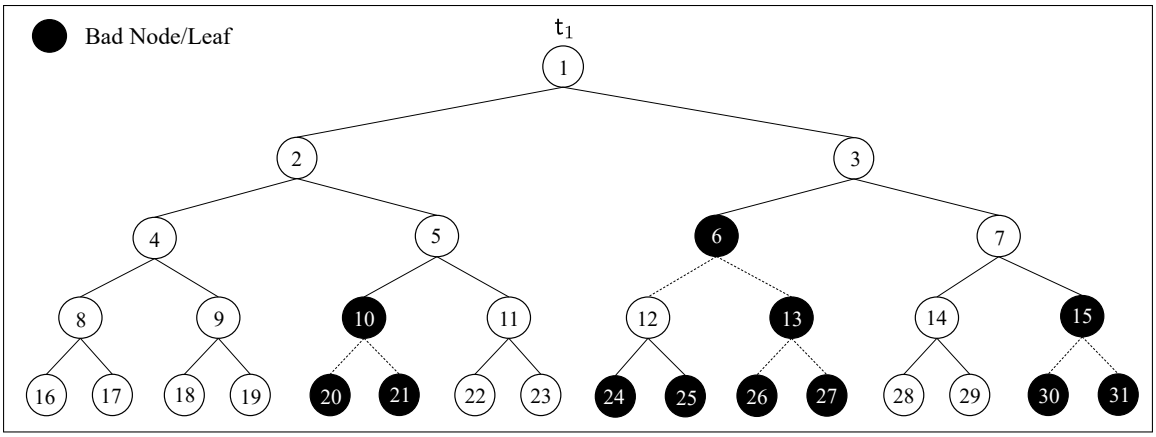

Fig. 1. A tree with 16 leaves, 8 Bad leaves, and 4 Bad (non-leaf) nodes. Solid lines indicate consistent hash relation and dashed lines indicate inconsistent hash relation.

constant and $t = \omega(\log \lambda)$. This implies that $\mathsf{t}_1 = \mathsf{t}_\mathsf{x}$ must happen with overwhelming probability. Therefore, the following holds:

$$\begin{aligned}
&\Pr\left[(Y \neq \bot) \wedge \mathsf{Bad}_{\mathsf{t}_1}\right] \\
&= \Pr\left[(Y \neq \bot) \wedge \mathsf{Bad}_{\mathsf{t}_1} \wedge (\mathsf{t}_1 = \mathsf{t}_\mathsf{x})\right] + \Pr\left[(Y \neq \bot) \wedge \mathsf{Bad}_{\mathsf{t}_1} \wedge (\mathsf{t}_1 \neq \mathsf{t}_\mathsf{x})\right] \\
&= \Pr[\mathsf{Bad}_{\mathsf{t}_1} \wedge (\mathsf{t}_1 = \mathsf{t}_\mathsf{x})] \cdot \Pr\left[(Y \neq \bot) \;\middle|\; \mathsf{Bad}_{\mathsf{t}_1} \wedge (\mathsf{t}_1 = \mathsf{t}_\mathsf{x})\right] + \mathsf{negl}(\lambda),
\end{aligned}$$

where the probability is taken over the same random procedure as in Claim 10. To prove Claim 10, it now suffices to show the following

$$\Pr\left[(Y \neq \bot) \;\middle|\; \mathsf{Bad}_{\mathsf{t}_1} \wedge (\mathsf{t}_1 = \mathsf{t}_\mathsf{x})\right] \leq \mathsf{negl}(\lambda).$$

Note that $Y \neq \bot$ represents the event that R does not abort *until the end of the protocol*. Therefore, we only need to prove that

$$\Pr\left[R \text{ does not abort until Stage 5 (inclusively)} \;\middle|\; \mathsf{Bad}_{\mathsf{t}_1} \wedge \mathsf{t}_1 = \mathsf{t}_\mathsf{x}\right] \leq \mathsf{negl}(\lambda). \tag{39}$$

Assuming that $\mathsf{Bad}_{\mathsf{t}_1}$ happens and that $\mathsf{t}_1 = \mathsf{t}_\mathsf{x}$, R does not abort in Stage 5 only if its challenge $\mathsf{r} = (b_1, \ldots, b_t)$ (in Stage 3) hits the leaves that have sibling paths consistent with the root t_1 $(= \mathsf{t}_\mathsf{x})$. Since t_x is fixed before r (a random subset of $[n]$), this event happens with probability $< (1-\delta)^t$, which is negligible as $0 < \delta < 1$ is a constant and $t = \omega(\lambda)$. Therefore, Inequality (39) holds.

This finishes the proof for Claim 10. □

In the following, we show the existence of a preimage for Y given that $Y \neq \bot$ and that $\mathsf{Bad}_{\mathsf{t}_1}$ does not happen. Note that Y can be parsed as

$$Y = \mathsf{t}_1 \| \mathsf{t}_2 \| (\mathsf{P}_{b_1}, \ldots, \mathsf{P}_{b_t}) \| (b_1, \ldots, b_t),$$

where $(\mathsf{P}_{b_1}, \ldots, \mathsf{P}_{b_t})$ are consistent with t_1. Since $\mathsf{Bad}_{\mathsf{t}_1}$ does not happen, there is a perfect binary tree consistent with the root t_1 that has at most δn "Bad" leaves. On the path from the root t_1 to each Bad leaf, there exists a node of minimal depth (i.e. closest to the root) for which the hash consistency breaks (i.e. this node is not equal to the hash value of the concatenation of its two children). We call such nodes Bad nodes. Note that the number of Bad nodes cannot exceed δn. (See Fig. 1 for an example.) W.l.o.g., we assume that there are exact $k = \delta n$ Bad nodes, and denote their indices in the tree as $\{p_1, \ldots, p_k\}$. We refer to the content of node p_i as v_{p_i} for all $i \in [k]$.

Figure 4: Functionality $\mathcal{F}_{F,\phi}$ for Proof-Based OWFs Supporting Predicates

The ideal functionality $\mathcal{F}_F$ interacts with a sender S and a receiver R. Upon receiving the input $\mathsf{x} \in \{0,1\}^{a(\lambda)}$ from S and $\mathsf{r} \in \{0,1\}^{b(\lambda)}$ from R, the functionality $\mathcal{F}_F$ sends $\mathsf{x}\|\mathsf{r}$ to S, and $(F(\mathsf{x}\|\mathsf{r}), \phi(\alpha))$ to R, where α is the prefix of x with the length satisfying ϕ's input requirement.

Now we describe how to find a preimage $X = \mathsf{x}\|\mathsf{r}$ for Y. We set r as the $(b_1, \ldots, b_t)$ part in Y. The x_i's in x are defined in the following way: if the i-th leaf in the above binary tree is not Bad, set x_i to be the contents of this leaf; otherwise, set x_i to be a dummy string (e.g. 0^λ). The (p_i, v_{p_i})'s consist of the indices and corresponding contents Bad nodes. We set τ such that it triggers the **Editing** condition in Constr. 2 (i.e. $\tau \neq z$ and $h(\tau) = h(z)$)[12]. And μ is set to the value t_2 in Y.

To see why the above $\mathsf{x}\|\mathsf{r}$ is a valid preimage for Y under $H_z^{h_i}$, just follow the evaluation procedure in Constr. 2. The **Editing** condition is triggered due to the way we set τ and the fact that the set of Bad indices $\{p_1, \ldots, p_k\}$ does not overlap with $\mathsf{Ind}(b_1, \ldots, b_t)$ (i.e. the indices of nodes on the sibling paths of leaves $\{x_{b_1}, \ldots, x_{b_t}\}$). Therefore, when building the Merkle tree, we will place v_{p_i} once we reach node p_i. Since all the non-Bad nodes in this Merkle tree are identical to the above binary tree, they necessarily share the same root value t_1. Finally, t_2 value will also be put in the correct position in Y as $\mathsf{t}_2 = \mu$ in the **Editing** case. □

Lemma 7 (Zero-Knowledge). *Prot. 2 satisfies the zero-knowledge requirement defined in Def. 14.*

Proof (Sketch). This lemma follows from an argument similar to that for Lem. 4. The ideal-world simulator Sim use S_1 (the **Commit**-stage simulator for Π_{ZKCnP}) and S_2 (the **Prove**-stage simulator for Π_{ZKCnP}) to go through Stage 2 and Stage 6 respectively. Note that Sim receives R's challenge r in Stage 3. It sends this r to the ideal functionality to learn Y. Similar as in the proof of Lem. 4, this Y value contains all the information to finish the remainder of the simulated interaction with R. □

8 Proof-Based One-Way Functions Supporting Predicates

Following the approach discussed in Sec. 2.4, we now describe how to extend our PB-OWF construction from Sec. 5 to support a predicate.

8.1 Definition

Definition 15 (PB-OWFs Supporting Predicates). *Let $a(\lambda)$, $b(\lambda)$ and $c(\lambda)$ be polynomials on λ. Let ϕ be an efficiently computable predicate. A* proof-based one-way function supporting ϕ, *denoted as (F, Π_ϕ), consists of a function $F_\lambda : \{0,1\}^{a(\lambda)} \times \{0,1\}^{b(\lambda)} \to \{0,1\}^{c(\lambda)}$ and a protocol $\Pi_\phi = \langle S, R\rangle_\phi$ of a pair of* PPT *machines. The function F satisfies the following one-wayness requirement:*

- **One-Wayness.** *F_λ satisfies the same one-way requirement as in Def. 12. That is, for all $\mathsf{r} \in \{0,1\}^{b(\lambda)}$, $F_\lambda(\cdot, \mathsf{r})$ is one-way.*

In the protocol Π, both parties take the security parameter (which we omit henceforth for simplicity) and a predicate ϕ as public input. The private input for S and R are x and r, respectively. At the end of the protocol, S outputs X, and R outputs $Y\|u$. The protocol satisfies the following requirements:

- **Completeness.** *The protocol Π computes the ideal functionality $\mathcal{F}_F$ defined in Fig. 4. Namely, $\forall \mathsf{x} \in \{0,1\}^{a(\lambda)}$ and $\forall \mathsf{r} \in \{0,1\}^{b(\lambda)}$, if $(X, Y\|u) \leftarrow \langle S(\mathsf{x}), R(\mathsf{r})\rangle_\phi$, then $X = \mathsf{x}\|\mathsf{r}$, $Y = F_\lambda(X)$ and $u = \phi(\alpha)$.*

[12] Note that such a τ exists with overwhelming probability. Because h_i maps length-$2m$ strings to length-m ones, it cannot be injective on more than 2^m strings in its domain. Since we pick $z \leftarrow \{0,1\}^{2m}$, with probability at least $(1 - \frac{1}{2^m})$, there exists a $\tau \neq z$ s.t. $h(z) = h(\tau)$.

– **Soundness.** *For every* PPT *machine* S^* *and every auxiliary input* $z \in \{0,1\}^*$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}\mathsf{r} \leftarrow \{0,1\}^{b(\lambda)};\\ (\cdot, Y \| u) \leftarrow \langle S^*(z), R(\mathsf{r})\rangle_\phi\end{array} : \begin{array}{l}Y \neq \bot \textbf{ and}\\ \nexists \mathsf{x} \text{ s.t. } (F_\lambda(\mathsf{x}\|\mathsf{r}) = Y \wedge \phi(\alpha) = u)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random sampling of r*, and the randomness used by* S^* *and* R*.*

– **Zero-Knowledge.** *This property is defined in the same way as the ZK property of Def. 12, but w.r.t. the ideal functionality specified in Fig. 4.*

8.2 Our Construction

To show how we can modify the constructions in Sec. 5 to satisfy the above requirements, we first review Constr. 1 and Prot. 1 from a new perspective.

A New Interpretation of Our PB-OWF Construction. Recall that the input to F^f in Constr. 1 takes the following form:

$$\mathsf{x} = \underbrace{(x_1, \ldots, x_n)}_{\alpha} \| \underbrace{(p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k})}_{\beta}, \text{ and } \mathsf{r} = (b_1, \ldots, b_t).$$

On an input $\mathsf{x} = \alpha\|\beta\|\mathsf{r}$, we can think of Constr. 1 as applying the following 3 steps:

– **Encoding:** apply some encoding Enc on α. In Constr. 1, $\mathsf{Enc}(\alpha)$ simply partitions α to x_i's;

– **Hardness-Inducing:** perform some one-way operation on $\mathsf{Enc}(\alpha)$ to ensure that the output is hard to invert. In Constr. 1, this one-way operation is just applying the oracle OWF f to each element of $\mathsf{Enc}(\alpha) = (x_1, \ldots, x_n)$. In Prot. 1, the sender reveals some portion of $\mathsf{Enc}(\alpha)$ specified by the receiver's challenge r. These openings allow R to check if S performs honestly. However, this can only ensure that S^* does not cheat for a large portion of $\mathsf{Enc}(\alpha)$; there is still a small part (say, a δ fraction) of $\mathsf{Enc}(\alpha)$ on which S^* can cheat. Therefore, we need the following **Editing** step;

– **Editing:** when β and r satisfy some predefined condition, we will edit the output of last step using the values contained in β. This step is critical to ensure the existence of a preimage under F^f for the Y learned by the receiver: as mentioned in last step, there is a small portion of $\mathsf{Enc}(\alpha)$ on which S^* may cheat. This **Editing** step essentially extends the pre-image set of Y from a single $\mathsf{Enc}(\alpha)$ to all the strings within a δ fractional distance to the valid $\mathsf{Enc}(\alpha)$. In this way, $Y \neq \bot$ will always have a preimage even if S^* can cheat on a δ fraction of $\mathsf{Enc}(\alpha)$ without being caught by R.

In light of the above perspective, we now discuss how to make the construction support ZK proofs for an arbitrary predicate $\phi(\cdot)$. Very roughly, we will pick a more robust **Encoding** approach that binds the sender to a unique α after the execution of the **Compute** stage. Moreover, it should be flexible enough to allow the sender to prove that this α satisfies some predicate $\phi(\cdot)$ later in the **Prove** stage. The **Hardness-Inducing** step will also need some change to accommodate our new **Encoding** method (such that we do not hurt the one-wayness). In the following, we elaborate on our construction.

The New Function F^f. We use a (n,t)-perfectly secure verifiable secret sharing VSS as our new Enc. Note that VSS is a randomized procedure. Fortunately, x is a random string. So we will draw randomness from some part (denoted as η) of x. That is, we encode α as $\mathsf{Enc}(\alpha;\eta) \coloneqq \mathsf{VSS}_{\mathsf{Share}}(\alpha;\eta) = ([\alpha]_1, \ldots, [\alpha]_n)$, where $\{[\alpha]_i\}$ are the VSS shares. Now, we want to apply some **Hardness-Inducing** on $\mathsf{Enc}(\alpha;\eta)$. Note that it does not suffice anymore to apply the oracle OWF f on these shares (like what we did in Constr. 1), because these VSS shares are correlated (thus, α may be recoverable from $\{f([\alpha]_i)\}_{i\in[n]}$). Therefore, we instead apply Naor's commitment to these shares, which can be built from OWF in black-box. Note that Naor's commitment also requires randomness, which will be obtained from other parts of x. For the **Editing** step, we use the same approach as in Constr. 1. We formalize the above intuition by showing the complete description of our new F^f in Constr. 3.

The New Protocol Π^f. The protocol that computes our new F^f follows the same template as Prot. 1. The formal description is given in Prot. 3. We now explain it by comparing it with Prot. 1. The R in Prot. 3 additionally takes a string ρ as input. Looking ahead, this ρ will be used as the first message for Naor's commitment. The sender parses its input x as:

$$\mathsf{x} = \alpha \| \eta \| \underbrace{(\gamma_1, \ldots, \gamma_n)}_{\gamma} \| (p_1, c'_{p_1}), \ldots, (p_k, c'_{p_k}).$$

Same as before, S first encodes α using a (n,t)-perfectly secure VSS scheme with randomness η. Denote the resulting shares as $\{[\alpha]_i\}_{i\in[n]}$. S then commits to these shares (in parallel) using Naor's commitment, using ρ as the first message and $\{\gamma\}_{i\in[n]}$ as the respective randomness. That is, S define the following values:

$$c_1 = \mathsf{Com}_\rho([\alpha]_1; \gamma_1), \ldots, c_n = \mathsf{Com}_\rho([\alpha]_n; \gamma_n).$$

Note that S has not sent to R these values yet. These $\{c_i\}_{i\in[n]}$ values should be viewed as the analogs of $\{y_i\}_{i=[n]}$ in Stage 1 of Prot. 1. One can think of them as the result of our new **Encoding** and **Hardness-Inducing** performed on the α part in x.

Recall that the protocol Π^f needs to additionally let R learn $\phi(\alpha)$. To do that, we use the "MPC-in-the-head" technique [IKOS07, GLOV12]. The sender emulates "in his head" n parties $\{P_i\}_{i\in[n]}$, where P_i's input is the i-th share $[\alpha]_i$. These parties execute a (n,t)-perfectly secure MPC protocol for computing $\phi(\alpha)$. Let $\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ denote the views of the n parties during the MPC execution. The sender appends these views to the ν committed in BBCom, and later reveals those specified by receiver's r (see below). From now on, Prot. 3 proceeds in an identical way as Prot. 1 except for the "consistency-checking" step:

- in Stage 5 of Prot. 1, R only checks $y_i = f(x_i)$ for the (x_i, y_i) pairs revealed by S according to r;
- while in Stage 7 of Prot. 3, the revealed $\{c_{b_i}, [\alpha]_{b_i}, \mathsf{v}_{b_i}\}_{i\in[t]}$ values (according to r) have a slightly more complex relation due to our new **Encoding** and **Hardness-Inducing** method. Here, R needs to verify that c_{b_i} is indeed a valid Naor's commitment to $[\alpha]_{b_i}$, and that the revealed $\{[\alpha]_{b_i}, \mathsf{v}_{b_i}\}_{i\in[t]}$ values are consistent MPC inputs and views for the corresponding parties. This is crucial for us to obtain soundness in this setting (proved formally in Lem. 9).

Other parts of the **Compute** phase (especially, how the **Editing** is handled) are done in the same way as Prot. 1. The ZK property can be proved as before, plus the (n,t)-privacy of the VSS and MPC protocol.

Formal Description. We present the formal construction in Constr. 3 and Prot. 3, relying on the following building blocks (in addition to the f and Π_{ZKCnP} for the construction in Sec. 5).

- Naor's commitment scheme Com [Nao90], which is a two-round statistically-binding commitment making only black-box use of f.
- $(n+1,t)$-perfectly secure VSS scheme $\mathsf{VSS} = (\mathsf{VSS}_{\mathsf{Share}}, \mathsf{VSS}_{\mathsf{Recon}})$ (see Def. 2);
- A (n,t)-perfectly secure MPC protocol (see Def. 3 and Rmk. 2);

For the VSS and MPC protocols, we require that t is a constant fraction of n such that $t \le n/3$. There are known constructions satisfying these properties [BGW88, CDD+99].

8.3 Security Proof

Theorem 6. *The constructions shown in Constr. 3 and Prot. 3 is a PB-OWF that supports predicates (as per Def. 15). Moreover, it only makes black-box use of OWFs.*

It follows immediately from the description that our construction makes only black-box use of the OWF f, and it also satisfies the complement requirement as per Def. 15. In the following, we establish the one-wayness in Lem. 8, soundness in Lem. 9, and zero-knowledge property in Lem. 10.

Lemma 8 (One-Wayness). *The function F^f in Constr. 3 is one-way as defined in Def. 15.*

Construction 3: The New One-Way Function $F^f(\cdot)$

Let $n(\lambda)$ be a polynomial on λ. Let both t and k be a constant fraction of n such that $k < t \leq n/3$. On input X, $F^f(X)$ is be computed as follows:

1. Parse the input X as the following two parts:

$$\mathsf{x} = \alpha \| \eta \| \gamma \| (p_1, c'_{p_1}), \ldots, (p_k, c'_{p_k}), \text{ and } \mathsf{r} \| \rho \text{ where } \mathsf{r} = (b_1, \ldots, b_t),$$

   where $|\alpha| = \lambda$, $|\rho| = 3\lambda$, and $\{p_i\}_{i\in[k]}$ and $\{b_i\}_{i\in[t]}$ should be long enough such that they form a size-k subset and a size-t subset of $[n]$, respectively (also see Rmk. 4).

   Remark 6. (On the length of η, γ and c'_{p_i}'s) The η should be long enough such that it can be used as the random tape in the VSS execution. The γ should be long enough such that it can be used as the randomness used in Step 3 to commit to the VSS shares of α. Each c'_{p_i} is of the same length as a commitment to a VSS share (i.e., the length of c_i in Step 3). Given a concrete VSS scheme, the length of these values can be determined accordingly.

2. Emulate $n+1$ (virtual) players $\{P_i\}_{i\in[n+1]}$ to execute the VSS protocol, where the input to P_{n+1} (i.e., the Dealer) is α. At the end of the execution, each player P_i ($i \in [n]$) obtains a share of α denoted as $[\alpha]_i$. The randomness for this part is taken from η.
3. For $i \in [n]$, apply Naor's commitment Com on each $[\alpha]_i$, where the first message for Com is set to ρ, and the randomness used by the committer is from $\gamma = (\gamma_1, \ldots, \gamma_n)$. Namely, it generates:

$$c_1 = \mathsf{Com}_\rho([\alpha]_1; \gamma_1), \ldots, c_n = \mathsf{Com}_\rho([\alpha]_n; \gamma_n).$$

4. Compute $(s_1, \ldots, s_n)$ as follows:
   (a) if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then let $s_i := c_i$ for all $i \in [n]$.
   (b) if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then let $s_i := \begin{cases} c'_i & i \in \{p_1, \ldots, p_k\} \\ c_i & i \in [n] \setminus \{p_1, \ldots, p_k\} \end{cases}$.
5. Output $Y = (s_1, \ldots, s_n) \| ([\alpha]_{b_1}, \gamma_{b_1}), \ldots, ([\alpha]_{b_t}, \gamma_{b_t}) \| \rho \| (b_1, \ldots, b_t)$.

Proof. We reduce one-wayness to the computationally-hiding property of Naor's commitment Com. Concretely, given a PPT machine $\mathcal{A}_{\text{OW}}$ breaking one-wayness of F^f, we will construct a PPT machine $\mathcal{A}_{\mathsf{Com}}$ that breaks the security of Com. The machine $\mathcal{A}_{\mathsf{Com}}$ works in the following way:

- It samples randomly the $\mathsf{x} = \alpha \| \eta \| \gamma \| (p_1, c'_{p_1}), \ldots, (p_k, c'_{p_k})$, $\mathsf{r} = \{b_1, \ldots, b_t\}$ and ρ defined in Constr. 3.
- It computes $\{[\alpha]_1, \ldots, [\alpha]_n\}$ as specified in Step 2 of Constr. 3.
- $\mathcal{A}_{\mathsf{Com}}$ forwards ρ (the first-round message for Naor's commitment) and $\overline{m}_0 = \{0^{|[\alpha]_i|}\}_{i\in[n]\setminus\mathsf{r}}$ and $\overline{m}_1 = \{[\alpha]_i\}_{i\in[n]\setminus\mathsf{r}}$ to the Com challenger.
- The challenger picks a random bit $b \leftarrow \{0,1\}$ and commits to each element in $\overline{m}_b$ in parallel, using ρ as the first message. Denote the commitments from the challenger as $\{c^*_i\}_{i\in[n]\setminus\mathsf{r}}$.
- $\mathcal{A}_{\mathsf{Com}}$ then computes $\{s_i\}_{i\in[n]}$ where $s_i = \begin{cases} c^*_i & i \in [n] \setminus \mathsf{r} \\ \mathsf{Com}_\rho([\alpha]_i; \gamma_i) & i \in \mathsf{r} \end{cases}$.
- It internally invokes $\mathcal{A}_{\text{OW}}$ on the input

$$Y^* := (s_1, \ldots, s_n) \| ([\alpha]_{b_1}, \gamma_{b_1}), \ldots, ([\alpha]_{b_t}, \gamma_{b_t}) \| \rho \| (b_1, \ldots, b_t),$$

  and in turn receives a value X^*.

Protocol 3: Protocol $\Pi^f_{F,\phi}$

Input: the security parameter 1^λ is the common input. The sender S takes x as its private input; the receiver R takes (r, ρ) as its private input.

1. R sends ρ to S;
2. S parses the input as $\mathsf{x} = \alpha \| \eta \| \gamma \| (p_1, c'_{p_1}), \ldots, (p_k, c'_{p_k})$. S computes the $(c_1, \ldots, c_n)$ as defined in Step 3 of Constr. 3. Note that $\{c_i\}_{i\in[n]}$ are supposed to be the commitments to the VSS shares $\{[\alpha]_i\}_{i\in[n]}$.
3. S emulates in its head n (virtual) players $\{P_i\}_{i\in[n]}$, where P_i's input is $[\alpha]_i$. These n parties execute the the (n,t)-perfectly secure MPC protocol for the following functionality: the functionality reconstructs α from $\{[\alpha]_i\}_{i\in[n]}$ (collected from each party), and sends $\phi(\alpha)$ to all the parties as their output. For $i \in [n]$, let v_i be the view of party P_i during the MPC execution.
4. S and R executes $\mathsf{BBCom}(\nu)$, the **Commit** stage of Π_{ZKCnP}, where S commits to the value
$$\nu \coloneqq (c_1, \ldots, c_n) \| (\mathsf{v}_1, \ldots, \mathsf{v}_n) \| (p_1, c'_{p_1}), \ldots, (p_k, c'_{p_k}). \tag{40}$$
5. R sends r to S.
6. S interprets r as a size-t subset $\{b_1, \ldots, b_t\} \subseteq [n]$. S computes the $\mathsf{s} = (s_1, \ldots, s_n)$ as defined in Step 4 of Constr. 3. S sends to R the values s and $\{(c_{b_i}, [\alpha]_{b_i}, \gamma_{b_i}, \mathsf{v}_{b_i})\}_{i\in[t]}$. (Recall that $[\alpha]_{b_i}$ is the value committed in c_{b_i} using randomness γ_{b_i}.)
7. Upon receiving the values S sends in last stage, R checks:
   (a) $\mathsf{Com}_\rho([\alpha]_{b_i}; \gamma_{b_i}) = c_{b_i}$ holds for all $i \in [t]$; **and**
   (b) $\{[\alpha]_{b_1}, \ldots, [\alpha]_{b_t}\}$ are consistent w.r.t. the VSS procedure; **and**
   (c) $\{\mathsf{v}_{b_1}, \ldots, \mathsf{v}_{b_t}\}$ constitute consistent (as per Def. 4) views w.r.t. the MPC execution as described in Stage 3. We remark that this includes checking that $[\alpha]_{b_i}$ is the prefix of v_{b_i}.

   If all the checks pass, R proceeds to next step; otherwise, R halts and outputs $\perp$.
8. S and R execute $\mathsf{BBProve}$, the **Prove** stage of Π_{ZKCnP}, where S proves that it performed Stage 6 honestly. That is, S proves that the ν committed at Stage 4 satisfies the following conditions:
   (a) the values $\{p_1, \ldots, p_k\}$ contained in ν form a size-k subset of $[n]$; **and**
   (b) the $(c_{b_1}, \ldots, c_{b_t})$ revealed by S in Stage 6 do consist of the subset of $\{c_i\}_{i\in[n]}$ (contained in ν) specified by r (sent by R in Stage 5); **and**
   (c) the $(\mathsf{v}_{b_1}, \ldots, \mathsf{v}_{b_t})$ revealed by S in Stage 6 do consist of the subset of $\{\mathsf{v}_i\}_{i\in[n]}$ specified by r; **and**
   (d) The $\mathsf{s} = (s_1, \ldots, s_n)$ satisfies the following conditions:
   - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then $s_i = c_i$ for all $i \in [n]$.
   - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then $s_i = \begin{cases} c'_i & i \in \{p_1, \ldots, p_k\} \\ c_i & i \in [n] \setminus \{p_1, \ldots, p_k\} \end{cases}$.
9. **(Receiver's Output.)** R outputs $Y = (s_1, \ldots, s_n) \| ([\alpha]_{b_1}, \gamma_{b_1}), \ldots, ([\alpha]_{b_t}, \gamma_{b_t}) \| \rho \| (b_1, \ldots, b_t)$ and $u = \phi(\alpha)$ (this value can be obtained from the MPC views revealed to R).
10. **(Sender's Output.)** S outputs $X = \alpha \| \eta \| \gamma \| (p_1, c'_{p_1}), \ldots, (p_k, c'_{p_k}) \| \rho \| (b_1, \ldots, b_t)$.

– $\mathcal{A}_{\mathsf{Com}}$ outputs 1 if and only if $F^f(X^*) = Y^*$.

We then argue that $\mathcal{A}_{\mathsf{Com}}$ wins the hiding game with non-negligible probability. There are two possible cases depending on the Com challenger's choice of b:

1. the challenger commits to $\overline{m}_1$: the Y^* is identically distributed as a $F^f(\cdot)$ evaluation on a random input. Thus, $\mathcal{A}_{\mathrm{ow}}$ will find a valid preimage X^* with non-negligible probability, which implies that $\mathcal{A}_{\mathsf{Com}}$ will

guess $b = 1$ correctly with non-negligible probability. We remark that there is a negligible chance that $\mathsf{r} \cap \{p_1, \ldots, p_k\} \neq \emptyset$. But this can be safely ignored without affecting the current proof.

2. the challenger commits to $\overline{m}_0$: note that $\overline{m}_0 \cup \{\mathsf{v}_i\}_{i\in[n]\setminus\mathsf{r}}$ cannot be consistent VSS shares as $\overline{m}_0$ contains only 0 strings. Moreover, it follows from the statistically-binding property that $\{c^*_{b_1}\}_{i\in[t]}$ (the commitments to $\overline{m}_0$) cannot be interpreted as commitments to values other than $\overline{m}_0$ (except with negligible probability). Since these $\{c^*_{b_1}\}_{i\in[t]}$ values are contained in Y^*, even an unbounded adversary cannot find and X^* such that $F^f(X^*) = Y^*$ (except with negligible probability). Thus, in this case, $\mathcal{A}_{\mathsf{Com}}$ will output 1 with negligible probability.

The above analysis shows that $\left|\Pr[\mathcal{A}_{\mathsf{Com}} = 1|b = 1] - Pr[\mathcal{A}_{\mathsf{Com}} = 1|b = 0]\right|$ is non-negligible. Therefore, $\mathcal{A}_{\mathsf{Com}}$ breaks the hiding property of Com. □

Lemma 9 (Soundness.). *Prot. 3 satisfies the soundness property specified in Def. 15.*

Proof. From our new perspective described at the beginning of Sec. 8.2, it is not hard to see that the soundness follows from the same argument used for Lem. 3 except for the following two caveats:

1. As mentioned earlier, the **Encoding** and **Hardness-Inducing** methods used in Prot. 3 (and Constr. 3) are different from those in Prot. 1. As a result, the receiver in Prot. 3 needs to check the consistency of the revealed VSS shares. We need to argue that such checks suffice to ensure the existence of a preimage.
2. Lem. 3 ensures the existence of a preimage X for the non-aborting Y learned by R. In Prot. 3, R additionally output a value u; so we need to (additionally) argue that u is the result of ϕ evaluated on the prefix α of X.

In the following, we address the above two points in order.

Addressing the First Point. To prove soundness for the current construction, we need to argue formally that if S^* cheats on the $(c_1, \ldots, c_n)$ values, R will abort with overwhelming probability. Recall that the counterpart of c_i's in Prot. 1 are $(y_1, \ldots, y_n)$, where y_i is supposed to be $f(x_i)$ for some underlying x_i; and in the proof of Lem. 3, we need to prove Claim 9, which says that there are at most δn many "bad" y_i's (i.e., there are no pre-images for them). In the current proof, we need an analog of this claim: here, the c_i's are commitments to VSS shares; so we need to argue that if S^* commits to more than k "bad" shares (defined later), R will abort with overwhelming probability by checking t (out of n) shares.

Formally, let $([\alpha]^*_1, \ldots, [\alpha]^*_n)$ be the values committed in $\{c_i\}_{i\in[n]}$ by S^*. We denote the following event, which should be considered as an analog of the event "Bad_Y" defined in the proof of Lem. 3:

– Bad_Y: there are at least $k = \delta n$ shares in $\{[\alpha]^*_i\}_{i\in[n]}$ that are "bad", i.e., it is *impossible* to convert these $\{[\alpha]^*_i\}_{i\in[n]}$ values to consistent VSS shares by modifying less than $k = \delta n$ of them (where $0 < \delta < 1$ is a constant).

We now show that, by checking a size-t random subset of them, R will catch at least one *pair* of inconsistent shares with overwhelming probability. In the following, we prove it relying on the "inconsistency graph" technique from [IKOS07].

Define a graph G with n vertices (corresponding to the n views). Assign an edge between node i and j in G if the views $[\alpha]^*_i$ and $[\alpha]^*_j$ are inconsistent w.r.t. VSS. When Bad_Y happens, it must be that the *minimum vertex cover* of G has size at least $k = \delta n$. We would like to argue that a random choice of t (a constant fraction of n) vertices will hit an edge with overwhelming probability. For this, we use the well-known connection between the size of a minimum vertex cover to the size of a *maximum matching*. Concretely, the graph G must have a matching $\mathcal{M}$ of size at least $\frac{\delta n}{2}$. (Otherwise, if the maximum matching contains less than $\frac{\delta n}{2}$ edges, the vertices of this matching form a vertex cover of size less than δn.) If the receiver R picks at least one edge of G, he will reject. The probability that the t vertices (shares) that the receiver picks miss all the edges of G is smaller than the probability that he misses all edges in $\mathcal{M}$. As shown in the following, the latter is negligible.

We use P to denote the number of distinct pairs of vertices in G, i.e. $P := \binom{n}{2}$. We use T to denote the distinct pairs of t different vertices, i.e. $T := \binom{t}{2}$. We use K to denote the size of $\mathcal{M}$, i.e. $K := \frac{\delta n}{2}$. Then, the

following holds:

$$\begin{aligned}\Pr[\text{all edges in } \mathcal{M} \text{ are missed}] &= \frac{\binom{P-K}{T}}{\binom{P}{T}} = \frac{(P-K-T+1)(P-K-T+2)\cdots(P-K)}{(P-T+1)(P-T+2)\cdots P} \\ &= \left(1-\frac{K}{P-T+1}\right)\left(1-\frac{K}{P-T+2}\right)\cdots\left(1-\frac{K}{P}\right) \\ &\leq \left(1-\frac{K}{P}\right)^{T} = \left(1-\frac{\delta}{n-1}\right)^{\frac{t(t-1)}{2}} \\ &= e^{-\Omega(\frac{t^2}{n})} \end{aligned} \tag{41}$$

By our choice of parameters, t is a constant fraction of n. Therefore, the above probability is negligible.

Remark 7 (Binding to α). It is worth noting that the above argument shows that there are at least $(n-k)$ consistent VSS shares. Since the parameter k is no larger than the VSS threshold t (by our choice of parameter), these $(n-k)$ consistent shares statistically bind the sender to a unique α.

Addressing the Second Point. To handle this issue, Prot. 3 follows the black-box commit-and-prove technique in [IKOS07, GLOV12, GOSV14]: in Stage 3, the sender performs the MPC-in-the-head execution to compute $\phi(\alpha)$; later in Stage 5 and Stage 7c, the receiver checks a size-t random subset of the views of the MPC-in-the-head parties. Following the same "inconsistency graph" argument as before, it is not hard to see that at least $(n-k)$ parties have consistent views w.r.t. the MPC execution for $\phi(\alpha)$. Since $k \leq t$ and the MPC protocol used in our construction is (n,t)-perfectly secure[13], it follows that $\phi(\alpha)$ is computed honestly. More accurately, this means that at least $(n-k)$ parties receive the same $\phi(\alpha)$ value as the output, where the α is the (unique) value reconstructed from the VSS shares from these $(n-k)$ parties' input (i.e., the value to which S was bound as described in Rmk. 7).

This finishes the proof for soundness. □

Lemma 10 (Zero-Knowledge). *Prot. 3 satisfies the zero-knowledge property defined in Def. 15.*

Proof. To prove the zero-knowledge property, we need to show a PPT ideal-world simulator Sim for any PPT malicious receiver R^*. At a high-level, such a simulator can be constructed as follows. Sim will use the simulator of the commit-and-prove protocol Π_{ZKCnP} at Stages 4 and 8. We will show how this allows Sim to finish the interaction without knowing the sender's input x.

Formally, we will build 2 hybrids starting from the real execution between the honest sender and R^*, and show that the second hybrid is essentially the simulator we want. We use Out_{H_i} $(i \in \{0,1\})$ to denote the output of hybrid H_i.

Hybrid $H_0(1^\lambda, \mathsf{x}, z)$. This hybrid uses the strategy of the honest $S(1^\lambda, \mathsf{x})$ to interact with the corrupted receiver $R^*(1^\lambda, z)$. At the end of the execution, H_0 outputs whatever R^* outputs. This hybrid is exactly the real execution.

Hybrid $H_1(1^\lambda, \mathsf{x}, z)$. This hybrid is identical to the previous one, except that

- At Stage 4, instead of executing BBCom, H_1 uses the strategy of Sim_1 in its communication with R^*, where Sim_1 is the simulator for the **Commit** stage of Π_{ZKCnP} (see Def. 8).
- At Stage 8, instead of doing the proof honestly, H_1 uses the strategy of Sim_2 in its communication with R^*, where Sim_1 is the simulator for the **Prove** stage of Π_{ZKCnP} (see Def. 8).

$\mathsf{Out}_{H_0} \overset{c}{\approx} \mathsf{Out}_{H_1}$: This is due to the ZK property of Π_{ZKCnP}.

The Simulator $\mathsf{Sim}(1^\lambda, z)$. We now describe the simulator Sim for the ideal execution. Sim is identical to H_1 except for the following changes:

[13] Here, (n,t)-perfect robustness (Def. 7) will suffice.

– Sim does not need to execute Stage 2;

– Upon receiving $\mathsf{r} = (b_1, \ldots, b_t)$ at Stage 5, Sim sends r and ρ (received from Stage 1) to the idea functionality $\mathcal{F}_{F,\phi}$, and receives back

$$Y = (s_1, \ldots, s_n) \| ([\alpha]_{b_1}, \gamma_{b_1}), \ldots, ([\alpha]_{b_t}, \gamma_{b_t}) \| \rho \| (b_1, \ldots, b_t), \quad \text{and} \quad \phi(\alpha).$$

– With $\{[\alpha]_{b_i}\}_{i\in[t]}$ (the input to P_{b_i}'s) and $\phi(x)$ (the output of P_{b_i}'s), Sim run the MPC simulator to generate the simulated views $\{\tilde{\mathsf{v}}_{b_i}\}_{i\in[t]}$ for parties $\{P_{b_i}\}_{i\in[t]}$. Since the MPC protocol is (n,t)-perfectly secure[14], the simulated views are identically distributed to the views in the real execution.

– At Stage 6, Sim sends to R^* the values $\mathsf{s} = (s_1, \ldots, s_n)$ and $\{(c_{b_i}, [\alpha]_{b_i}, \gamma_{b_i}, \tilde{\mathsf{v}}_{b_i})\}_{i\in[t]}$ (Note that s and $\{(c_{b_i}, [\alpha]_{b_i}, \gamma_{b_i})\}_{i\in[t]}$ are contained in Y^*.)

We remark that, unlike H_1, Sim does not need to know the input x to the sender; the Y^* it obtained from $\mathcal{F}_{F,\phi}$ contains all the information it needs to finish its execution with R^*. In particular, although the s and $\{(c_{b_i}, [\alpha]_{b_i}, \gamma_{b_i})\}_{i\in[t]}$ values now come from the ideal functionality $\mathcal{F}_{F,\phi}$, they are identically distributed to the ones generated by the honest sender. As mentioned earlier, the simulated views $\{\tilde{\mathsf{v}}_{b_i}\}_{i\in[t]}$ are also identically distributed to the views in the real execution. It then follows that the output of Sim is identical to H_1.

This finishes the proof for Lem. 10. □

9 Acknowledgments

We thank the anonymous Crypto 2021 reviewers for their detailed reviews and valuable comments.

[14] Here, semi-honest (n,t)-computational privacy (Def. 5) will suffice.

References

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press. 11, 12, 40

BM17. Boaz Barak and Mohammad Mahmoody-Ghidary. Merkle's key agreement protocol is optimal: An $O(n^2)$ attack on any key agreement from random oracles. *Journal of Cryptology*, 30(3):699–734, July 2017. 14

CDD+99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany. 11, 40

CGMA85. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press. 11

CK88. Claude Crépeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 42–52, White Plains, NY, USA, October 24–26, 1988. IEEE Computer Society Press. 3

CLP20. Rohit Chatterjee, Xiao Liang, and Omkant Pandey. Improved black-box constructions of composable secure computation. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020: 47th International Colloquium on Automata, Languages and Programming*, volume 168 of *LIPIcs*, pages 28:1–28:20, Saarbrücken, Germany, July 8–11, 2020. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. 3, 13

Coo71. Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971. 3

Dam90. Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. 34

DI05. Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. 3

FS90. Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd Annual ACM Symposium on Theory of Computing*, pages 416–426, Baltimore, MD, USA, May 14–16, 1990. ACM Press. 3

GGMP16. Sanjam Garg, Divya Gupta, Peihan Miao, and Omkant Pandey. Secure multiparty RAM computation in constant rounds. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 491–520, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. 3

GIKR01. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd Annual ACM Symposium on Theory of Computing*, pages 580–589, Crete, Greece, July 6–8, 2001. ACM Press. 11

GKP18. Sanjam Garg, Susumu Kiyoshima, and Omkant Pandey. A new approach to black-box concurrent secure computation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 566–599, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 3

GL89. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press. 4

GLM+04. Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 258–277, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 3

GLOV12. Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd Annual Symposium on Foundations of Computer Science*, pages 51–60, New Brunswick, NJ, USA, October 20–23, 2012. IEEE Computer Society Press. 3, 5, 13, 40, 44

GLPV20. Sanjam Garg, Xiao Liang, Omkant Pandey, and Ivan Visconti. Black-box constructions of bounded-concurrent secure computation. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 87–107, Amalfi, Italy, September 14–16, 2020. Springer, Heidelberg, Germany. 3

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press. 3

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 Annual Symposium on Foundations of Computer Science*, pages 174–187, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. 3

Gol01. Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001. 10

Gol04. Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004. 10, 12, 24

GOSV14. Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 515–524, New York, NY, USA, May 31 – June 3, 2014. ACM Press. 3, 44

Goy11. Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 695–704, San Jose, CA, USA, June 6–8, 2011. ACM Press. 3

Hai08. Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 412–426, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany. 3

HILL99. Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. 4

HR04. Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 92–105, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany. 10

HV16. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On the power of secure two-party computation. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 397–429, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. 3

HV18. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Round-optimal fully black-box zero-knowledge arguments from one-way permutations. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 263–285, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. 3

IKLP06. Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing*, pages 99–108, Seattle, WA, USA, May 21–23, 2006. ACM Press. 3

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30, San Diego, CA, USA, June 11–13, 2007. ACM Press. 3, 6, 13, 40, 43, 44

ILL89. Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st Annual ACM Symposium on Theory of Computing*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press. 4

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture*

Notes in Computer Science, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany. 3

IR89. Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press. 4, 14, 17

Kar72. Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. 3

Kil88. Joe Kilian. Founding cryptography on oblivious transfer. In *20th Annual ACM Symposium on Theory of Computing*, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press. 3

Kiy14. Susumu Kiyoshima. Round-efficient black-box construction of composable multi-party computation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 351–368, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. 3

Kiy20. Susumu Kiyoshima. Round-optimal black-box commit-and-prove with succinct communication. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 533–561, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. 3

KOS18. Dakshita Khurana, Rafail Ostrovsky, and Akshayaram Srinivasan. Round optimal black-box "commit-and-prove". In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 286–313, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. 3, 5

Lev73. Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. 3

LP12. Huijia Lin and Rafael Pass. Black-box constructions of composable protocols without set-up. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 461–478, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 3

Mer90. Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. 33, 34

Nao90. Moni Naor. Bit commitment using pseudo-randomness. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 128–136, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. 10, 40

PW09. Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 403–418. Springer, Heidelberg, Germany, March 15–17, 2009. 3

Ros12. Mike Rosulek. Must you know the code of f to securely compute f? In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 87–104, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 3, 4, 6, 15

RTV04. Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 4, 14

Wee10. Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st Annual Symposium on Foundations of Computer Science*, pages 531–540, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press. 3

Wyn75. Aaron D Wyner. The wire-tap channel. *Bell system technical journal*, 54(8):1355–1387, 1975. 3

Yer11. Arkady Boris Yerukhimovich. *A study of separations in cryptography: new results and new models*. PhD thesis, University of Maryland, College Park, Maryland, USA, 2011. 17

Supplementary Material

A Proof-Based Pseudo-Random Generators (Full Version)

A.1 Definition

Definition 16 (Proof-Based PRGs). *Let $a(\lambda)$, $b(\lambda)$ and $c(\lambda)$ be polynomials on λ. A proof-based pseudorandom generator consists of function $G_\lambda : \{0,1\}^{a(\lambda)+b(\lambda)} \to \{0,1\}^{c(\lambda)}$ and a protocol $\Pi = (S, R)$ involving a pair of* PPT *machines. We use $(X, Y) \leftarrow \langle S(1^\lambda, \mathsf{x}), R(1^\lambda, \mathsf{r})\rangle$ to denote the execution of protocol Π where the security parameter is λ, the inputs to S and R are x and r respectively, and the outputs of S and R are X and Y respectively. Let $Y = \perp$ denote that R aborts in the execution. The following conditions hold:*

- **Pseudo-randomness.** *For every $\mathsf{r} \in \{0,1\}^{b(\lambda)}$, $G_\lambda(\cdot\|\mathsf{r})$ is a PRG on its first input. That is, it is efficiently computable, length stretching, and*

$$\{\mathsf{x} \leftarrow \{0,1\}^{a(\lambda)} : G_\lambda(\mathsf{x}\|\mathsf{r})\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{U_{c(\lambda)}\}_{\lambda\in\mathbb{N}}.$$

- **Completeness.** *$\forall\lambda \in \mathbb{N}$, $\forall\mathsf{x} \in \{0,1\}^{a(\lambda)}$ and $\forall\mathsf{r} \in \{0,1\}^{b(\lambda)}$, if $(X, Y) \leftarrow \langle S(1^\lambda, \mathsf{x}), R(1^\lambda, \mathsf{r})\rangle$, then $X = \mathsf{x}\|\mathsf{r}$ and $Y = G_\lambda(\mathsf{x}\|\mathsf{r})$.*

- **Soundness.** *For every* PPT *machine S^* and every auxiliary input $z \in \{0,1\}^*$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\begin{array}{l}\mathsf{r} \leftarrow \{0,1\}^{b(\lambda)};\\ (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle\end{array} : \begin{array}{l}Y \neq \perp \textbf{ and}\\ \nexists\mathsf{x} \text{ s.t. } G_\lambda(\mathsf{x}\|\mathsf{r}) = Y\end{array}\right] \leq \mathsf{negl}(\lambda)$$

  where the probability is taken over the random sampling of r,and the randomness used by S^* and R.

- **Zero-Knowledge.** *This property is defined by requiring the security against corrupted R in the ideal-real paradigm for 2PC w.r.t. the ideal functionality $\mathcal{F}_G$, which is obtained by replacing F with G in Fig. 1. Namely, there exist a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$, $\forall\mathsf{x} \in \{0,1\}^{a(\lambda)}$, $\forall\mathsf{r} \in \{0,1\}^{b(\lambda)}$, and $\forall z \in \{0,1\}^*$,*

$$\left\{\mathsf{REAL}_{\Pi,\mathcal{A}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})\right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{IDEAL}_{\mathcal{F}_G,\mathsf{Sim}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})\right\}_{\lambda\in\mathbb{N}}.$$

  where $\mathsf{REAL}_{\Pi,\mathcal{A}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})$ and $\mathsf{IDEAL}_{\mathcal{F}_G,\mathsf{Sim}(z)}(1^\lambda, \mathsf{x}, \mathsf{r})$ are defined in the same way as in Def. 12.

A.2 Our Construction

We now present our construction for proof-based PRGs, thus establishing the following theorem.

Theorem 7. *There exists a PB-PRG that satisfies Def. 16 and makes only black-box use of PRGs.*

Our construction consists of a PRG G^g (Constr. 4) together with a black-box protocol (Prot. 4) that proves the membership for G^g. The construction relies on the following building blocks:

- A pseudo-random generator g;
- A zero-knowledge commit-and-prove protocol $\Pi_{\mathsf{ZKCnP}} = (\mathsf{BBCom}, \mathsf{BBProve})$ as per Def. 8. Such protocols can be constructed assuming only black-box access to g.

It follows immediately from the description that our construction makes only black-box access to PRGs. In Appx. A.3, we show that it satisfies definition Def. 16.

Construction 4: Pseudo-Random Generator G^g

Let $m(\lambda)$ and $n(\lambda)$ be polynomials on λ. Let $0 < \delta < 1$ be a constant, and $k(\lambda) = \delta n(\lambda)$. Let $t(\lambda) = \log^2(\lambda)$. Assume that $g : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ is a PRG. On input $\mathsf{x} \in \{0,1\}^{n\lambda + k(\log(n)+m)}$ and $\mathsf{r} \in \{0,1\}^{t\log(n)}$, G^g parses them as

$$\mathsf{x} = (x_1, \ldots, x_n) \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k}), \text{ and } \mathsf{r} = (b_1, \ldots, b_t),$$

where $|x_i| = \lambda$, $|y'_{p_i}| = m$, $\{p_i\}_{i\in[k]}$ is a size-k subset of $[n]$, and $\{b_i\}_{i\in[t]}$ is a size-t subset of $[n]$. G^g outputs $Y = (z_1, \ldots, z_n)$, which are computed as follows:

1. if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then $z_i = \begin{cases} x_i & i \in \{b_1, \ldots, b_t\} \\ g(x_i) & \text{otherwise} \end{cases}$;
2. if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then $z_i \coloneqq \begin{cases} x_i & i \in \{b_1, \ldots, b_t\} \\ y'_i & i \in \{p_1, \ldots, p_k\} \\ g(x_i) & \text{otherwise} \end{cases}$.

A.3 Proof of Security

In this section, we prove that the function G^g in Constr. 4 and Prot. 4 constitute a black-box PRG with proof (as per Def. 16). In Lem. 11, we prove that G^g satisfies the pseudo-randomness requirement in Def. 16. Given the description of Constr. 4, the completeness of Prot. 4 follows immediately by construction. We then establish the soundness and zero-knowledge property in Lem. 12 and 13 respectively.

Lemma 11 (Pseudo-randomness of G^g). *Constr. 4 satisfies the pseudo-randomness property defined in Def. 16.*

Proof (Sketch). We first argue that G^g is length-stretching. To see that, note that the input and output are of the following length respectively:

$$|X| = |\mathsf{x}| + |\mathsf{r}| = \delta nm + \delta n \log(n) + \lambda n + t\log(n), \text{ and } |Y| = nm - tm + \lambda t,$$

where $0 < \delta < 1$ is a constant, $t = \log^2(\lambda)$ (see also Rmk. 4), and m and n are polynomials on λ. Note that we have control over choice of m and n. For example, If we set $n = \Omega(\lambda^2)$ and $|m| = \Omega(\log n)$, then the dominating term for the input length will be δmn, and the dominating term for the output length will be mn. Since $0 < \delta < 1$, G^g is length-stretching.

The pseudo-randomness of G^g follows from standard hybrid arguments. The only point that requires extra attention is that the input to G^g has two parts x and r, and we need to prove the pseudo-randomness of its output for *all* $\mathsf{r} \in \{0,1\}^{b(\lambda)}$. Note that since x is sampled randomly, Case 1 in Constr. 4 happens with overwhelming probability. Therefore, the pseudo-randomness of G^g (for all r) can be reduced to the pseudo-randomness of g by standard hybrid technique. We omit the details. □

Lemma 12 (Soundness). *Protocol Π_{PRGP} in Prot. 4 satisfies the soundness property defined in Def. 16. Namely, for every* PPT *machine S^* and every auxiliary input $z \in \{0,1\}^*$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\begin{matrix}\mathsf{r} \leftarrow \{0,1\}^{t\log(n)}; \\ (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle\end{matrix} : \begin{matrix} Y \neq \perp \textbf{ and} \\ \nexists \mathsf{x} \text{ s.t. } G^g(\mathsf{x}\|\mathsf{r}) = Y\end{matrix}\right] \leq \mathsf{negl}(\lambda). \tag{43}$$

Proof. Let us first define "non-trivial" S^*'s, which are the malicious provers that can make the honest receiver accept with non-negligible probability.

Definition 17 (Non-Trivial S^*). *A PPT machine S^* is* non-trivial *if there exists some auxiliary input $z \in \{0,1\}^*$ such that the following holds: there exits a polynomial $\mathsf{poly}(\cdot)$ such that for infinitely many $\lambda \in \mathbb{N}$,*

$$\Pr\left[\mathsf{r} \leftarrow \{0,1\}^{t\cdot\log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle : Y \neq \perp\right] \geq \frac{1}{\mathsf{poly}(\lambda)}. \tag{44}$$

Protocol 4: Protocol Π_G^g for Our Proof-Based Pseudorandom Generator

Let g, m, n, k and t be as in Constr. 4.

Input: both parties take the security parameter 1^λ as the common input. Sender S takes a string $\mathsf{x} \in \{0,1\}^{n\lambda+(\log(n)+m)k}$ as private input; receiver R takes a string $\mathsf{r} \in \{0,1\}^{t\cdot\log(n)}$ as private input.

1. S parses the input as $\mathsf{x} = (x_1, \ldots, x_n) \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k})$, where $|x_i| = \lambda$ for all $i \in [n]$, $|y'_{p_j}| = m$ for all $j \in [k]$, and $\{p_i\}_{i\in[k]}$ forms a size-k subset of $[n]$. S defines a $2 \times n$ matrix $M = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}$, where $y_i = g(x_i)$ for all $i \in [n]$.
2. S and R execute $\mathsf{BBCom}(\alpha)$, the **Commit** stage of Π_{ZKCnP}, where S commits to the value
$$\alpha \coloneqq M \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k}). \tag{42}$$
3. R sends r to S.
4. S interprets r as a size-t subset $\{b_1, \ldots, b_t\} \subseteq [n]$, and defines $M_{\mathsf{r}} = \begin{bmatrix} x_{b_1} & \cdots & x_{b_t} \\ y_{b_1} & \cdots & y_{b_t} \end{bmatrix}$, i.e. the columns of M specified by r. S also computes $(z_1, \ldots, z_n)$ in the way specified in Constr. 4. S sends to R the values M_{r} and $(z_1, \ldots, z_n)$.
5. With M_{r}, R checks (via its oracle access to $g(\cdot)$) if $g(x_{b_i}) = y_{b_i}$ holds for all $i \in [t]$. If all the checks pass, R proceeds to next step; otherwise, R halts and outputs $\perp$.
6. S and R execute $\mathsf{BBProve}$, the **Prove** stage of Π_{ZKCnP}, where S proves that it performs Stage 4 honestly. Namely, S proves that the α committed at Stage 2 satisfies the following conditions:
   (a) the values $\{p_1, \ldots, p_k\}$ contained in α form a size-k subset of $[n]$; **and**
   (b) the M_{r} does consist of the columns in M specified by r; **and**
   (c) the $(z_1, \ldots, z_t)$ satisfy the following conditions:
   - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$, then $z_i = \begin{cases} x_i & i \in \{b_1, \ldots, b_t\} \\ y_i & \text{otherwise} \end{cases}$;
   - if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$, then $z_i \coloneqq \begin{cases} x_i & i \in \{b_1, \ldots, b_t\} \\ y'_i & i \in \{p_1, \ldots, p_k\} \\ y_i & \text{otherwise} \end{cases}$.

   Similar as in Prot. 1, these conditions can be expressed a predicate on α.
7. **(Receiver's Output).** R outputs $Y = (z_1, \ldots, z_n)$.
8. **(Sender's Output).** S outputs $X = (x_1, \ldots, x_n) \| (p_1, y'_{p_1}), \ldots, (p_k, y'_{p_k}) \| (b_1, \ldots, b_t)$.

It is not hard to see that if a PPT machine $S^*(1^\lambda, z)$ is *not* non-trivial for all $z \in \{0,1\}^*$, then Inequality (43) holds immediately. Therefore, to prove Lem. 12, we only need to focus on the non-trivial S^*'s.

For any $(\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle$ such that $Y \neq \perp$, the Y can then be parsed as $(z_1, \ldots, z_n)$ such that $|z_i| = \lambda$ for $i \in \{b_1, \ldots, b_t\}$ and $|z_i| = m$ for $i \in [n] \setminus \{b_1, \ldots, b_t\}$. Recall that $\{b_1, \ldots, b_t\}$ is the size-t subset of $[n]$ specified by r. Given such a Y and r, we say that a z_i is "no-preimage" if it satisfies the following requirements:

- z_i is in the set $\{z_1, \ldots, z_n\} \setminus \{z_{b_1}, \ldots, z_{b_t}\}$; **and**
- there does not exist any $x \in \{0,1\}^\lambda$ such that $g(x) = z_i$

We then define the following event

– $\mathsf{Bad}_{Y,\mathsf{r}}$: there are more than $k = \delta n$ number of "no-preimage" z_i's in Y.

In the following, we present a claim. We will first show how to prove Lem. 12 assuming that Claim 11 holds, and then present its proof.

Claim 11. *For every non-trivial S^* with the $z \in \{0,1\}^*$ satisfying Inequality (44), there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\mathsf{r} \leftarrow \{0,1\}^{t\cdot\log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle : \mathsf{Bad}_{Y,\mathsf{r}} \;\middle|\; Y \neq \bot\right] \leq \mathsf{negl}(\lambda). \tag{45}$$

Consider the Y output by R from $\langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle$, where $S^*(1^\lambda, z)$ is non-trivial and $\mathsf{r} \leftarrow \{0,1\}^{t\log(n)}$. If $Y \neq \bot$, it must be of the form $(z_1, \ldots, z_n)$, where $|z_i| = \lambda$ for $i \in \{b_1, \ldots, b_t\}$ and $|z_i| = m$ for $i \in [n] \setminus \{b_1, \ldots, b_t\}$. Recall that $\{b_1, \ldots, b_t\}$ is the size-t subset of $[n]$ specified by r.

Assuming that Claim 11 holds, to finish the proof of Lem. 12, it suffices to show the following claim:

– Conditioned on $Y \neq \bot$, if $\mathsf{Bad}_{Y,\mathsf{r}}$ does not happen, then there exist an x and a r such that $G^g(\mathsf{x}\|\mathsf{r}) = Y$.

Conditioned on $Y \neq \bot$, Claim 11 implies that there are at most $k = \delta n$ no-preimage z_i's (except with negligible probability). In the following, we assume w.l.o.g. that there are exactly k no-preimage z_i's. We denote them as $\{z_{p_1}, \ldots, z_{p_k}\}$ (i.e. we denote the indices of these no-preimage z_i's by $\{p_1, \ldots, p_k\}$). Then, for each z_i where $i \in [n] \setminus \{p_1, \ldots, p_k\}$, this z_i must have (at least) one preimage under $g(\cdot)$. We denote an arbitrary preimage of such z_i as $g^{-1}(z_i)$. By definition, a no-preimage z_i is *not* in the set $\{z_{b_1}, \ldots, z_{b_t}\}$. Therefore, we must have that $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$.

With these observations, we show in the following how to construct x and r such that $G^g(\mathsf{x}\|\mathsf{r}) = (z_1, \ldots, z_n)$. At a high-level, we take advantage of Case 2. At a high-level, we will put those no-preimage z_i's together with their indices as the (p_i, y'_{p_i}) part of x. We will use the above $\mathsf{r} = \{b_1, \ldots, b_t\}$ from R. Since $\{b_1, \ldots, b_t\} \cap \{p_1, \ldots, p_k\} = \emptyset$, G^g will put the no-preimage z_{p_i}'s at position p_i's (are required by Case 2). This will give us the desired Y. Concretely, we set:

$$\mathsf{x} := (x'_1, \ldots, x'_n)\|(p_1, z_{p_1}), \ldots, (p_k, z_{p_k}), \quad \mathsf{r} := (b_1, \ldots, , b_t)$$

where x'_i's are defined as follows: $\forall i \in [n]$, $\quad x'_i := \begin{cases} z_i & i \in \{b_1, \ldots, b_t\} \\ 0^\lambda & i \in \{p_1, \ldots, p_k\} \\ g^{-1}(z_i) & \text{otherwise} \end{cases}$.

We remark that $g^{-1}(z_i)$ may not be efficiently computable (indeed, g is a PRG). But this is fine since we only require the existence of $g^{-1}(z_i)$. Also, note that $\{p_1 \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. It then follows from the description in Constr. 1 (in particular, Case 2) that $G^g(\mathsf{x}\|\mathsf{r}) = Y$.

In the following, we show the proof for Claim 11, which will complete the proof for Lem. 12.

Proof of Claim 11. All the probabilities appearing in this proof are taken over the following random procedure:

$$\mathsf{r} \leftarrow \{0,1\}^{t\cdot\log(n)}; (\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle,$$

where S^* and z are as described in Claim 11.

First, note that $\Pr\left[\mathsf{Bad}_{Y,\mathsf{r}} \;\middle|\; (Y \neq \bot)\right] \cdot \Pr[Y \neq \bot] = \Pr[\mathsf{Bad}_{Y,\mathsf{r}} \wedge (Y \neq \bot)]$. Since S^* is non-trivial, we know from Inequality (44) that $\Pr[Y \neq \bot]$ is non-negligible. Therefore, to prove Inequality (45), it suffices to show

$$\Pr[\mathsf{Bad}_{Y,\mathsf{r}} \wedge (Y \neq \bot)] \leq \mathsf{negl}(\lambda), \tag{46}$$

which we prove in the following.

Consider the execution $(\cdot, Y) \leftarrow \langle S^*(1^\lambda, z), R(1^\lambda, \mathsf{r})\rangle$ where $Y \neq \bot$. Observe that S^* at Stage 2 commits a value α[15] of the form shown in Expression (42). For this execution, we define the following sequence of events:

[15] This α is well-defined as BBCom is statistically-binding.

- E_1: $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} = \emptyset$. The $\{p_i\}_{i \in [k]}$ are those contained in α and $\{b_i\}_{i \in [t]}$ are those specified by R's input r.
- E_2: $\forall i \in [n] \setminus \{b_1, \ldots, b_t\}$, $y_i = z_i$. The y_i's are contained in the second row of M (which is in turn contained in α), and z_i's are contained in Y.
- E_3: the M_{r} sent by S^* at Stage 4 does consist of the columns of M specified by $\mathsf{r} = \{b_1, \ldots, b_t\}$.
- E_4: there are more than $k = \delta n$ number of y_i's (contained in the second row of M) that satisfy the following requirement: there does not exist any $x \in \{0,1\}^\lambda$ such that $g(x) = y_i$.

We first claim:

$$\Pr[E_4 \wedge (Y \neq \bot)] \leq \mathsf{negl}(\lambda) \tag{47}$$

To see that, first notice the folloing:

$$\begin{aligned}\Pr[E_4 \wedge (Y \neq \bot)] &= \Pr[E_4 \wedge (Y \neq \bot) \wedge E_3] + \Pr\left[E_4 \wedge (Y \neq \bot) \wedge \overline{E}_3\right] \\ &\leq \underbrace{\Pr\left[(Y \neq \bot) \,\middle|\, E_3 \wedge E_4\right]}_{\mathsf{P}_1} + \underbrace{\Pr\left[(Y \neq \bot) \wedge \overline{E}_3\right]}_{\mathsf{P}_2}\end{aligned}$$

We now show that both P_1 and P_2 are negligible. First, recall that R checks at Stage 5 if $y_i = g(x_i)$ for all columns $[x_i \; y_i]^T$ contained in M_{r}. Thus, conditioned on E_3, the receiver does not abort only if the set $\mathsf{r} = \{b_1, \ldots, b_t\}$ *does not* select any "bad" column $[x_i \; y_i]^T$ in M s.t. $y_i \neq f(x_i)$. Moreover, E_4 ensures that there are more than $k = \delta n$ such "bad" columns in M. Therefore, the probability P_1 is smaller than $(1-\delta)^t$, which is negligible as $0 < \delta < 1$ is a constant and t is $\omega(\log \lambda)$.

Also, observe that E_3 was actually proved at Stage 6 (as Item 6b) by S^* via the commit-and-prove protocol Π_{ZKCnP}. The soundness of Π_{ZKCnP} implies that P_2 is negligible.

Next, we make another claim:

$$\Pr\left[\mathsf{Bad}_{Y,\mathsf{r}} \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \leq \mathsf{negl}(\lambda) \tag{48}$$

To prove this inequality, notice the following:

$$\begin{aligned}\Pr\left[\mathsf{Bad}_{Y,\mathsf{r}} \wedge (Y \neq \bot) \wedge \overline{E}_2\right] &= \Pr\left[\mathsf{Bad}_{Y,\mathsf{r}} \wedge (Y \neq \bot) \wedge \overline{E}_2 \wedge E_1\right] + \Pr\left[\mathsf{Bad}_{Y,\mathsf{r}} \wedge (Y \neq \bot) \wedge \overline{E}_2 \wedge \overline{E}_1\right] \\ &\leq \underbrace{\Pr[E_1]}_{\mathsf{P}_3} + \underbrace{\Pr\left[(Y \neq \bot) \wedge \overline{E}_2 \,\middle|\, \overline{E}_1\right]}_{\mathsf{P}_4}\end{aligned}$$

We now show that both P_3 and P_4 are negligible. Recall that E_1 is the event that the set $\{p_1, \ldots, p_k\}$ contained in α does not overlap with $\mathsf{r} = \{b_1, \ldots, b_t\}$. First, note that the Π_{ZKCnP} proof at Stage 6 ensures that, except with negligible probability, the set $\{p_1, \ldots, p_k\}$ is a size-k subset of $[n]$ (i.e. Item 6a). Also, observe that the r is a size-t random subset of $[n]$ that is sampled *independently* of $\{p_1, \ldots, p_k\}$. Therefore, E_1 happens with probability $\leq (1-\delta)^t + \mathsf{negl}(\lambda)$, which is negligible.

According to Constr. 1, if $\{p_1, \ldots, p_k\} \cap \{b_1, \ldots, b_t\} \neq \emptyset$ (which is exactly the event $\overline{E}_1$), then it must hold that $y_i = z_i$ for all $i \in [n] \setminus \{b_1, \ldots, b_t\}$ (which is exactly E_2). Also, recall that this condition is enforced by the $\mathsf{BBProve}$ performed by S^* at Stage 6 (see Item 6c). The soundness of the Π_{ZKCnP} guarantees that, conditioned on $\overline{E}_1$, if E_2 does not hold, then R will abort with overwhelming probability. Therefore, P_4 is negligible.

Before proving Inequality (46), we need one more claim:

$$\Pr\left[\mathsf{Bad}_{Y,\mathsf{r}} \,\middle|\, E_2\right] \leq \Pr\left[E_4 \,\middle|\, E_2\right] \tag{49}$$

To prove the above inequality, recall that if $\mathsf{Bad}_{Y,\mathsf{r}}$ happens, then there are more than $k = \delta n$ many z_i's in $\{z_1, \ldots, z_n\} \setminus \{z_{b_1}, \ldots, z_{b_t}\}$ that do not have any preimage under $g(\cdot)$. Moreover, conditioned on E_2, we known that $z_i = y_i$ for all $i \in [n] \setminus \{b_1, \ldots, b_t\}$. In this case, $\mathsf{Bad}_{Y,\mathsf{r}}$ implies that there are more than δn many y_i's, among all the y_i's whose index lies in $[n] \setminus \{b_1, \ldots, b_t\}$, that do not have any preimage under $g(\cdot)$. Thus,

there are definitely more than $k = \delta n$ many y_i's (among all the y_i's contained in the second row of M) that do not have any preimage, which is exactly the event E_4. Therefore, conditioned on E_2, $\mathsf{Bad}_{Y,\mathsf{r}}$ implies E_4. This gives us Inequality (49).

We are now ready to derive Inequality (46):

$$\begin{aligned}
\Pr[\mathsf{Bad}_Y \wedge (Y \neq \bot)] &= \Pr[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge E_2] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \\
&\leq \Pr[E_4 \wedge (Y \neq \bot) \wedge E_2] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \qquad (50) \\
&\leq \Pr[E_4 \wedge (Y \neq \bot)] + \Pr\left[\mathsf{Bad}_Y \wedge (Y \neq \bot) \wedge \overline{E}_2\right] \\
&\leq \mathsf{negl}(\lambda) \qquad (51)
\end{aligned}$$

where Inequality (50) follows from Inequality (49). Also, note that Inequality (51) follows from Inequalities (47) and (48). This finishes the proof of Claim 11.

This finishes the proof for Lem. 12. □

Lemma 13 (Zero-Knowledge). *Protocol Π^g_G in Prot. 4 satisfies the zero-knowledge property defined in Def. 16.*

Proof. To prove the zero-knowledge property, we need to show a PPT ideal-world simulator Sim for any PPT malicious receiver R^*. At a high-level, such a simulator can be constructed as follows. Sim will use the simulator of the commit-and-prove protocol Π_{ZKCnP} at Stages 2 and 6. This allows Sim to finish the interaction without knowing the sender's input x.

Formally, we will show two hybrids H_0 and H_1, where the H_0 is the real execution between the honest sender and R^*, and show that H_1 is essentially the simulator we want. We use Out_{H_i} to denote the output of hybrid H_i.

Hybrid $H_0(1^\lambda, \mathsf{x}, z)$. This hybrid uses the strategy of the honest $S(1^\lambda, \mathsf{x})$ to interact with the corrupted receiver $R^*(1^\lambda, z)$. At the end of the execution, H_0 outputs whatever R^* outputs. This hybrid is exactly the real execution.

Hybrid $H_1(1^\lambda, \mathsf{x}, z)$. This hybrid is identical to the previous one, except that

- At Stage 2, instead of executing BBCom, H_1 uses the strategy of Sim_1 in its communication with R^*, where Sim_1 is the simulator for the **Commit** stage of Π_{ZKCnP} (see Def. 8).
- At Stage 6, instead of doing the proof honestly, H_1 uses the strategy of Sim_2 in its communication with R^*, where Sim_1 is the simulator for the **Prove** stage of Π_{ZKCnP} (see Def. 8).

$\mathsf{Out}_{H_0} \overset{\mathrm{c}}{\approx} \mathsf{Out}_{H_1}$: This is due to the ZK property of Π_{ZKCnP}.

The Simulator $\mathsf{Sim}(1^\lambda, z)$. We now describe the simulator Sim for the ideal execution. Sim is identical to H_1 except for the following changes:

- Sim does not need to execute Stage 1;
- Upon receiving $\mathsf{r} = (b_1, \ldots, b_t)$ at Stage 3, Sim sends it to the idea functionality $\mathcal{F}_{G^g}$, and receives back $Y = (z_1, \ldots, z_n)$.
- At Stage 4, Sim (with its oracle access to $g(\cdot)$) sets $\widetilde{M}_\mathsf{r} = \begin{bmatrix} z_{b_1} & \cdots & z_{b_t} \\ g(z_{b_1}) & \cdots & g(z_{b_t}) \end{bmatrix}$, and sends $\widetilde{M}_\mathsf{r}$ and $(z_1, \ldots, z_n)$ to R^*.

We remark that, unlike H_1, Sim does not need to know the input x to the sender; the values $(z_1, \ldots, z_n)$ it obtains from $\mathcal{F}_{G^g}$ contain all the information to finish its execution with R^*. In particular, although the z_i's now come from the ideal functionality, they are identically distributed to the ones generated by the honest sender. Thus, the $\widetilde{M}_\mathsf{r}$ is also identically distributed to the M_r in H_1. It Then follows that the output of Sim is identical to H_1.

This finishes the proof for Lem. 13. □