

How to instantiate Fiat-Shamir Provably and Practically?

Riddhi Ghosal ^{*}
Nokia Bell Labs

Aniket Kate [†]
Purdue University, Mysten Labs

Pratyay Mukherjee [‡]
Hashgraph, Indian Statistical Institute, Kolkata

Pratik Sarkar [§]
Obsidus Labs

Rohit Sinha [¶]
Hashgraph

Abstract

We propose a new variant of the Fiat–Shamir transformation where the Fiat-Shamir hash is instantiated by an external verifiable random function (VRF) service. We prove that soundness holds as long as the VRF secret key remains hidden from the prover, i.e., assuming no collusion between the prover and the VRF evaluator. Crucially, due to the public verifiability of VRF outputs, the VRF computation itself need not be trusted beyond the key-generation phase. This yields *the first* NIZK compiler in a non-idealized model that is *both practical and provably sound*.

Our transformation can be instantiated using extensively deployed blockchain-based VRF services. Moreover, it naturally supports distributed VRF services, allowing the no-collusion assumption to be relaxed: soundness holds even if the prover colludes with up to a threshold number of evaluators. In this setting, distributed key generation for the VRF further eliminates trust in the setup.

A distinctive feature of our approach is that the verification step is local and does not require accessing the VRF. This is particularly advantageous for recursive proof systems such as incrementally verifiable computation, where the verification circuit must be embedded within the circuit for which a proof needs to be generated. This enables construction of the *first* provable and practical recursive proof – bypassing well-known impossibilities in the random oracle model.

We demonstrate the real-world utility of our framework with an implementation for the GKR protocol.

^{*}riddhi.ghosal@nokia-bell-labs.com, This work was mostly done as an intern at Supra, while the author was a PhD student at UCLA.

[†]aniket@purdue.edu

[‡]pratyay85@gmail.com

[§]iampratiksarkar@gmail.com, This work was mostly done when the author was at Supra Research.

[¶]sinharo@gmail.com

Contents

1	Introduction	2
1.1	Our Contribution	3
1.2	Relevant discussion on our model	5
1.3	Related Work	7
2	Technical Overview	8
2.1	Background on vulnerability of Fiat-Shamir	8
2.2	Our Solution: Use external randomness	9
2.3	Proof Outline	11
2.4	Extending to non-relativized model and an application to recursive proofs:	12
3	Preliminaries	13
3.1	Relativized Sigma Proof with Pre-Processing	13
3.2	Simulatable Verifiable Random Function [CL07]	15
4	Stateful Simulatable VRFs	17
5	Fiat-Shamir in the relativized VRF oracle model	18
5.1	Fiat-Shamir transformation in the relativized VRF model	18
6	Constructions of Stateful and Simulatable VRF	42
6.1	Black-box construction of ssVRF from sVRF	42
7	Fiat-Shamir transformation in the VRF oracle model and application to Recursive Proofs	43
7.1	Application to Recursive Proofs	45
7.2	Provably instantiating a state-of-the-art construction for recursive argument	47
8	Implementation and Evaluation	49
9	References	51
A	Provably instantiating NOVA: Prelims	57

1 Introduction

The Fiat-Shamir (FS) transformation [FS86] was initially introduced as a tool to transform identification protocols to digital signatures. It has since evolved into a pivotal technique in modern cryptography to convert public-coin interactive protocols to non-interactive ones. Pointcheval and Stern [PS96] gave the first rigorous proof of the security of this transformation in the random oracle model (ROM) [BR94]. Briefly, in the context of three-round public coin interactive protocols (aka Sigma protocols), in this methodology, the verifier’s public-coin message is replaced with a random oracle evaluation of the prover’s first round message, thereby making it deterministic and getting rid of the need to have any interaction ^[1]. This technique provides provable soundness of the non-interactive system in ROM as long as the underlying Sigma protocol is sound. In real-world uses, the random oracle is typically instantiated with a cryptographic hash function with the hope that, as long as the hash function is “sufficiently complex”, the verifier’s message still remains unpredictable to a malicious prover. As this approach achieves concrete efficiency guarantees, it has been widely used in practice to construct several succinct non-interactive arguments (of knowledge)(SNARG/SNARK) (See [CY24, Section 14], [T⁺22, Section 5]), and recursive proofs like Incrementally Verifiable Computation (IVC) in practice [Hal, plo, KS22].

However, proving the security of the FS transformation for any construction when the random oracle is replaced with a concrete hash function has proven challenging, and remains an open problem even today. Furthermore, we now have several counter-examples demonstrating insecurity of this technique. Specifically, Canetti, Goldreich, and Halevi [CGH04] provided constructions of cryptographic primitives that are secure in the random oracle model but become totally insecure when the random oracle is instantiated with *any* concrete hash function. Barak [Bar01], and Goldwasser and Kalai [GK03] extended this further by constructing secure interactive protocols for which the FS transform with *any* hash function results in an insecure non-interactive protocol. More recently, Bartusek et al. [BBH⁺19] showed that standard constructions of hash-based proofs like [Mic00, CY21, Kil92] are insecure when the FS transformation is applied. These results indicate that any claimed soundness about a concrete instantiation of an FS transformation can be heuristic, at best.

Practical attack on Fiat-Shamir. Overall, while we have known that there exist protocols for which applying the Fiat-Shamir transform is insecure, all the above attacks were somewhat contrived and did not apply against “real-world protocols”. The state of affairs changed drastically recently after Khovratovich, Rothblum, and Soukhanov [KRS25] presented an attack (henceforth, KRS) against the FS transformation for the widely popular GKR protocol [GKR15], and in turn, potentially targeting several practically deployed proof systems [Pol24, ZGK⁺17, WTS⁺18, XZZ⁺19, ZLW⁺21]. This attack demonstrates that the theoretical vulnerabilities can manifest in real-world systems, casting doubt on the security of many deployed cryptographic systems. All of the above attacks, including the most recent one, follow a similar approach—they construct a proof where the circuit for which the proof is being generated computes the FS hash function itself. The core idea behind such an attack, also broadly known as *diagonalization attacks*, is that by computing the hash function, the circuit is able to correctly predict the verifier’s challenge message before it gets generated, and then exploit this capability to generate false proofs.

Avoidance strategies and their limitations. Of course, if we limit the scope of the proving circuits only to those that do not compute the FS hash function, then diagonalization attacks would not apply. However, for natural and practically relevant proof systems such as *recursive proofs for incrementally verifiable computations* [KST22] (IVC) and proof carrying data [BCMS20a], such circuits are unavoidable. Indeed, in the recursive step, the circuits must execute verifier circuit of the base proof system; and all existing practical NIZK verification algorithms necessarily compute the FS hash. Consequently, FS-based recursive proof system instantiations not only lack a security proof, but also they are conceivably susceptible to diagonalization

^[1]This, of course, can be generalized to multiple rounds.

attacks.

Despite these attacks, and a lack of a comprehensive solution, instantiating the Fiat-Shamir transformation heuristically in the ROM remains the most versatile, and one of the *only* known technique to achieve *practical* efficiency and therefore, is the most widely used in the real-world constructions. This raises the stakes significantly to design an alternative framework.

Fiat-Shamir with provable security. *Theoretically*, an alternative that guarantees provable security and removes the possibilities of the aforementioned attacks are a class of constructions that falls under the correlation intractable (CI) hash function-based framework [CCR16, KRR17, CCRR18, HL18, CCH⁺19, PS19, BKM20, CSW22, JJ21, HLR21, CJJ21, HJKS22, KLV23]. However, the downside of using CI hash is that all the currently known constructions use complex underlying tools, which take them out of scope for practical constructions.

Our focus. As it turns out, most constructions can *either be deployed practically, or can guarantee provable security*^[2], *but not both* – this leaves us wondering whether this tradeoff between efficiency and provable security is indeed a fundamental necessity. This creates a rather unsatisfactory situation, and makes it an imperative to ask the following question:

Is it possible to instantiate the Fiat-Shamir transformation in a way that simultaneously achieves both efficiency and provable security?

1.1 Our Contribution

In this work, we precisely address this question by presenting a construction that achieves *both the desired properties*. Specifically, we propose a solution in which the Fiat-Shamir hash is instantiated by a simulatable verifiable random function (VRF) [MRV99]. Concretely, our construction supports provable soundness as long as the VRF evaluation key remains hidden from the prover, i.e., assuming no collusion between the prover and the evaluator, and the VRF key-generation procedure is trusted^[3]. Our construction, with this additional trust assumption, addresses all aforementioned concerns while being *practical, and provably instantiable in real-world*.^[4] Let us elaborate:

- Our **core technical idea** is to replace the random oracle with a *simulatable* VRF (sVRF) [CL07, GHK⁺25].^[5] We formalize this using a new oracle model, where the non-interactive prover and the verifier are given *oracle access* to an sVRF. The prover invokes the sVRF oracle to receive an evaluation along with an sVRF proof attesting to the validity of the evaluation. The evaluation is treated as the verifier’s challenge, while the sVRF proof string is appended to the prover’s message. The rest of the proof generation closely resembles FS in ROM. The verification procedure, in addition to running the usual steps, also checks whether the VRF evaluation (i.e. the challenge) was correct using the VRF proof.
- From a **theoretical angle** our construction has several important benefits: (i) It supports circuits of arbitrary size—in contrast, the recent XFS transformation [AY25] only supports circuits of a priori

^[2]Throughout the paper, we often use the term “provable security” slightly narrowly to imply provable security with concrete instantiation of the hash functions, in a non-idealized setting.

^[3]We elaborate our result with these trust assumptions only for the sake of simplicity. These can be alleviated using a distributed setup. We include detailed discussion in a later section

^[4]Our work is an alternate approach to provably instantiate FS itself as opposed to introducing a new transformation (as done in [AY25]) to remove interaction.

^[5]A VRF ensures pseudorandomness of the evaluation holds as long as the proof is not known, whereas an sVRF guarantees pseudorandomness even when the proof is available

bounded size.^[6] (ii) Moreover, our construction is provably sound in the standard model, if the sVRF instantiation is in the standard model (e.g. one based on Dodis-Yamploskiy [DY05, CL07]). This is in contrast to standard FS or XFS where the proof of soundness only holds in idealized models.

- From an **application perspective**, our construction can be seen as a practical and provable instantiation of a random oracle that can be easily deployed using real-world VRF frameworks [GLOW21, GHK⁺25, Res23, Lab22]. Note that such a framework already exists in practice, as many blockchain ecosystems do rely on VRFs for a provably secure source of randomness. However, our prover needs to query an external VRF service (possibly over blockchains), incurring communication delay. Nevertheless, with concrete experiments (See Section 8) we show that the runtime of the prover remains practical.
- As mentioned earlier, if the VRF evaluator is a single entity, we need to rely on a trusted key-generation procedure plus assume no collusion between the prover and the evaluator. However, these **assumptions can be effectively mitigated** by using a distributed VRF [GLOW21, GHK⁺25, Res23]. Soundness would continue to hold provably even if the prover colludes with (upto) a threshold number of VRF evaluators. Moreover, to eliminate the trust on VRF key-generation, a distributed key generation (DKG), such as [Gro21, KMM⁺24] can be used in this setting.
- Finally, our construction has **major implications to recursive proofs**. This is because our verification process is *local and deterministic*, i.e., it does not need access to *any* oracle. Consequently, a recursive proof where the circuit includes the verification circuit does not need to access the sVRF. This is in contrast to proofs in the idealized models, such as FS or XFS, where the verifier is required to query the random oracle.^[7] This implies that the verification algorithm cannot be expressed efficiently as an arithmetic circuit. As a consequence, the concrete instantiations of such proof systems with existing transforms need to first instantiate the RO heuristically with a concrete hash function, and therefore remain vulnerable to diagonalization attacks. In fact, the lack of an efficient description for the verification is the primary reason why we *do not* have provably secure recursive proofs in the ROM, and there are several well known impossibility results [Val08, HAN23, BCG24].

On the other hand, in our construction, the verifier checks the validity of the challenge using the VRF verification procedure. This step is performed locally and does not need to access the sVRF oracle, and thus allowing an explicit and efficient description of the verification circuit. In fact, this asymmetry in the oracles access of prover and verifier puts forward a new paradigm of instantiating FS which could be interesting on its own. In Section 7.2 we demonstrate how to use our transformation with NOVA: a state-of-the-art folding scheme based IVC construction. NOVA relies on a folding scheme that is made non-interactive by applying the FS transformation to a public-coin sigma protocol in the ROM. We observe that the RO in their construction can be replaced by a simulatable VRF in a modular fashion. This, in turn, yields an efficient construction of IVC with provable security guarantees. We give more details in Sections 2.4 and Section 7.

This is also the reason why it does not suffice to replace the random oracle in the FS transformation with a standard pseudorandom function (PRF). A PRF serves as a concrete replacement for a RO in several applications. However, a PRF does not support any local and deterministic verification, therefore does not enable one of the most important implications of our result. We discuss this further at the end of Section 2

^[6]Recently proposed by Arnon and Yorgev [AY25], the extended Fiat-Shamir framework (XFS) also supports provable security, yet requires different trade-offs, such as support for bounded size circuits only, compared to our approach. Furthermore, the provable guarantees for XFS only hold in an idealized model. We refer to Section 1.3 for a comparison.

^[7]Specifically, the verifier has to query the oracle to reproduce the challenge.

In Table 1, we present a comparison of our work with some other well-known approaches to construct (succinct) non-interactive arguments.

Table 1: Comparing different approaches to construct non-interactive proofs.

Category	Random Oracle Model		CRS Model		VRF Oracle Model
	Fiat-Shamir in ROM (FS)	Extended Fiat-Shamir in ROM (XFS)	Correlation-Intractable Hash based	Knowledge-Assumption based	
Illustrative Examples	[GKR15, GWC19, Mic00, CDG ⁺ 25]	[AY25]	[CCH ⁺ 19, PS19, BKM20, HLR21, CJJ21]	[Gro16]	Our Work
Provable Security [†]	No	No	Yes	Yes	Yes
Efficiently implementable	Yes	Yes	No	Yes	Yes
Hardness Assumptions	–	–	Falsifiable	Non-Falsifiable	Falsifiable
Trust Assumptions	None	None	Trusted CRS setup (instance independent)	Needs instance dependent setup	Trusted VRF key-gen setup (instance independent) + No collusion between VRF evaluator and prover [‡]

[†] Provable security means provable security with concrete instantiation in the non-idealized setting.

[‡] The trust assumption can be reduced by using a distributed VRF [GHK⁺25].

1.2 Relevant discussion on our model

On our trust assumptions. Indeed, our approach deviates from the usual Fiat-Shamir paradigm, as the proof generation requires interaction with an external VRF service. However, as discussed previously, our focus is to construct a framework for non-interactive proofs that is efficiently implementable without sacrificing provable security. In this work, we provide such a solution by additionally assuming a non-colluding VRF evaluator. We note that, since the VRF output is publicly verifiable, the computation itself *does not* need to be trusted beyond the key-generation setup. We believe, given the state of art, this is a reasonable trade off. In fact, this is in line with deployed VRF-based randomness services [GHK⁺25], where the unpredictability of randomness relies on secrecy of VRF evaluation key. While it is true that adding additional trust assumptions makes it relatively easier to achieve provable security, our community lacks a good understanding of what is the *minimal* extent of trust that is necessary. Indeed, recall that, there are well-known impossibility results regarding constructing succinct non-interactive proofs without using some form of a trusted setup (e.g. a common random/reference string) – the soundness crucially relies on the honest generation of the reference string, and non-collusion between the setup algorithm and prover.

Our setting may be thought of as an extension of this. Indeed, our assumption is stronger because not only do we have a trusted VRF setup (which happens once and is independent of the instance), but the VRF must also be evaluated during online proof generation (and therefore, is instance-dependent).^[8] To reduce trust, multiparty ceremony protocols [NR22] have been proposed in the literature. Moreover, there are numerous examples in real-world, where third parties are trusted with *entire proof generation*, for instance, outsourced proof services like Succinct, Nexus, and many more [Lab26, MAG⁺24]. In comparison, our assumption is rather mild.

^[8]For our idealized setting, we also need a trusted way to maintain a public state. In practice this is instantiated using, e.g. smart contracts [TX24, Con23, HMW18] supported by consensus.

In the worst case, one may wonder what happens if our trust model does not hold, and the VRF evaluation key is known to the prover. Here, we again stress that we *do not* trust the VRF service to perform a correct computation, because the VRF output can be verified publicly. And, exactly due to this, even if the VRF key is known to the prover, our approach seems to fall back to standard FS heuristic for “natural” VRF instantiations [GLOW21] that satisfy some form of “unbiasability”.^[9] The VRF evaluation can be thought of as an implementation of a “good” FS hash while the VRF output is verified during verification. So, the diagonalization attacks will apply again, though breaking soundness otherwise would be non-trivial. To summarize, even if the trust assumption does not hold, we are not worse off than standard FS.

Practical perspectives. We also point out that a VRF oracle model is natural from a practical perspective, as there are several VRF-based frameworks [GHK⁺25] deployed in the blockchain space that provide this service in a black-box manner [Lab22, Res23, Ora23, TX24]. These randomness services fulfill several millions of requests every months [Raq25]. In fact, more relevant to us, many of the deployed VRFs (e.g., Supra’s dVRF [GHK⁺25] and Aptos Roll [TX24]) already satisfy our requirement of ssVRF simply by maintaining a counter via smart-contract or assigning a unique block number and transaction ID to every query as a counter. We believe that a fundamental appeal of our work is to attract practitioners to use this already-existing infrastructure to produce provably sound proofs. Moreover, some of them [Res23, TX24] do use distributed VRFs [GLOW21], thereby readily support the opportunity to reduce trust.

Security against diagonalization attacks. Recall that the KRS attack, or more broadly, any diagonalization attack, crucially relies on the circuit’s ability to compute the same hash function as the FS hash. Consequently, any countermeasure must somehow restrict the circuit from computing that. An elegant solution was proposed in a recent work by Arnon and Yogev [AY25] – the construction compels the prover to solve a Proof-of-Work (POW) puzzle. It evades the KRS attack by putting a size bound on the circuit, so that the POW component can not be computed as a part of circuit execution. Instead, we rely on an external VRF service, which holds a secret key unknown to the circuit. To formally show the security of our transformation, we follow an approach similar to [AY25], and propose a new relativized VRF oracle model where we allow the interactive prover, and verifier to have access to this oracle. We provide the model description and formal soundness proofs in Section 3 and 5 respectively. This model is motivated by the one proposed in [AY25] and we intend it to use it as a much needed concrete theoretical evidence that attacks such as the one proposed by [KRS25] is not possible in our model. Since our analysis focuses only for three-round Sigma protocols, similar to [AY25], it does not provably provide immunity against the KRS [KRS25] attack because it inherently has a multi-round proof, but it provides strong evidence that using our technique to the GKR protocol makes the attack infeasible. Extending our result to the multi-round setting remains an open problem, and we leave it for future work.

The construction in the relativized VRF model is more involved and requires an enhanced notion of a stateful and simulatable VRF (ssVRF). Specifically, we need to use a simulatable VRF whose evaluation function maintains a local state that gets updated to a unique value after every invocation. We refer to Section 4 for a formal definition. Moreover, we show in Section 6 that several known constructions of simulatable VRF can be slightly modified to satisfy this notion. Our result (cf. Section 7) about the (non-relativized) Fiat-Shamir transformation in the simulatable VRF oracle model follows almost as a corollary from this result.

On (non)-interactiveness. Finally, one can observe that our construction does allow some form of interaction, since the prover can query the VRF service during the proof generation phase. So, it is natural to question if we are constructing a non-interactive proof after all. We emphasize that our construction does guarantee a meaningful form of non-interactivity because our construction still supports a *single, one-shot*

^[9]Unbiasability would guarantee that the VRF output, though predictable, would look like random, as long as the secret key is correctly generated. There are contrived VRFs which would fail to satisfy this, but for existing constructions this property is likely to hold, heuristically.

communication from the prover to the verifier. While the prover is allowed to interact with the VRF service, there is no requirement for the verifier to remain online (or even to be known to the prover) during the proof generation to respond to the prover. Moreover, our verification process is local, so the verifier does not need to have any interaction with the VRF service (or any other external entity). This makes our proof publicly verifiable – a crucial property that distinguishes non-interactive proofs from interactive ones. Additionally, in most applications, it is natural for a proof to be generated once, but the verification is run several times by different users. Our construction can be effectively used in such a setting, unlike an interactive proof that would require trust and communication among verifiers or regeneration of the proof for every individual verifier.

1.3 Related Work

We covered most related works along the line of Fiat-Shamir above. Here we provide details of two very relevant ones.

Extended Fiat-Shamir (XFS) [AY25]. Arnon and Yogev [AY25] neatly point out the core reason that creates vulnerabilities in the FS transformation when the random oracle is replaced with a cryptographic hash function. Specifically, *in the random oracle model, the circuit is incapable of performing certain operations (for instance, compute the random oracle on any arbitrary input) that are absolutely feasible in the real-world with any concrete hash function* – the standard FS soundness proof crucially relies on the fact that the circuit does not have the ability to compute the FS hash.^[10] The KRS attack exploits this gap by providing a circuit that is able to compute the exact hash function used for FS, and thereby predicts the challenge. So, to capture this, they [AY25] use a relativized model [CT10, CCS22, CCG⁺23, CGSY24, BCG24], in which all entities, including the interactive prover, the interactive verifier and the circuit, are allowed to have access to the random oracle. In this model, one can have circuit that predicts verifier’s challenge by querying the random oracle. So, the standard FS transformation’s soundness proof in the ROM no longer holds, and in fact, the KRS attack extends easily to this model. In this model they [AY25] then propose an augmented FS transformation called Extended Fiat-Shamir (XFS), which is shown to be provably secure in the relativized random oracle model. The XFS transformation essentially combines the standard Fiat-Shamir with a new type of proof-of-work (POW) [DN93]. They prove soundness in the relativized model and therefore provide credible evidence that “diagonalization” attacks are unlikely to be successful in the standard model, if XFS is used instead of standard Fiat-Shamir. While this approach takes a crucial first step towards enhancing the security of Fiat-Shamir, this approach has certain important shortcomings:

- (i) XFS support proofs only for a bounded-sized circuits. This is because the circuit must not be able to produce a proof of work on a random instance. Thus, fixing the hardness of the POW imposes an upper bound on the size of the circuit in order to prevent it from solving the POW instance.
- (ii) XFS can only be proven secure in the random oracle model, which means that we are back to heuristic security guarantees when instantiating this in the real world. Indeed, in order to prove soundness, the POW instance and subsequently, the verifier’s challenge must be generated via a random oracle.

As already detailed, we improve on all these fronts. In addition to an analysis in the idealized setting (which is similar to [AY25]), unlike [AY25], our result is also meaningful in the standard non-relativized setting, or just the VRF oracle model, which, in turn, can be instantiated without random oracles – this is discussed in Section 7.

Groth16. Another alternative proof system is Groth16 [Gro16] that is both provably secure and practically efficient. However, in Groth16, every proof needs to have a fresh trusted-setup, which is a significant

^[10]Even if the circuit has some hash function hardcoded, that must be different from the hash used for FS.

downside. Additionally, from a theoretical standpoint, the assumptions used in Groth16 are *non-falsifiable*, and therefore, not well-understood.

Recursive proofs without random oracle. Plonk is one of the most influential and widely used zk-SNARKs widely used in recursive proofs. Recently, Fucksbauer and Sefranek [IAC26] showed that for analyzing Plonk, the ROM can be avoided. However, their proofs are in the Algebraic Group Model – another idealized model.

2 Technical Overview

We will first give a brief background about vulnerabilities of the Fiat-Shamir transformation [KRS25] that will add more context to our approach. Then, we will proceed with detailing our primary contribution.

2.1 Background on vulnerability of Fiat-Shamir

The following is an illustrative toy Sigma protocol that effectively captures the known attacks on the Fiat-Shamir transformation. This protocol, inspired by Barak [Bar01] has been borrowed verbatim from [AY25]. We present it here for the sake of completeness and to provide some context to the reader.

In this protocol, a prover provides a circuit $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^{r+1}$ which is preprocessed into a digest ψ and an input/output pair $(\mathbf{x}, \mathbf{y})$, and claims to have a witness $\mathbf{w}$ so that $C(\mathbf{x}, \mathbf{w}) = \mathbf{y}$. Specifically, the protocol works as follows:

1. **(Preprocessing):** The preprocessing algorithm receives a circuit C and outputs a short digest ψ of this circuit. The honest prover is given C , and the verifier is given ψ .^[11]
2. **(Prover):** The honest prover sends a succinct commitment of the witness $m_1 = \text{Com}(\mathbf{w})$ to the verifier.
3. **(Verifier):** The verifier replies with a random challenge $\beta \leftarrow \{0, 1\}^r$.
4. **(Prover):** The honest prover sends C and $\mathbf{w}$ to the verifier.
5. **(Verifier’s decision):** The verifier accepts if ψ is the digest of C , it holds that $m_1 = \text{Com}(\mathbf{w})$, and either $C(\mathbf{x}, \mathbf{w}) = \mathbf{y}$, or $C(\mathbf{x}, \mathbf{w}) = (0\|\beta)$.

This is a trivial protocol where the prover sends C and $\mathbf{w}$ to the verifier. In real-world instantiations, this step is replaced by more sophisticated techniques.

If there indeed exists a witness $\mathbf{w}$ so that $C(\mathbf{x}, \mathbf{w}) = \mathbf{y}$, then the honest prover will send this, and it will be accepted by the verifier. Furthermore, the protocol has small soundness error because a malicious prover that chooses C , $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{w}$ cannot predict the verifier’s challenge β as it is sampled uniformly at random after the witness has been committed to. Essentially, the second acceptance condition will never be satisfied. However, as we will see, this protocol is not secure after applying the heuristic instantiation of the Fiat-Shamir transformation.

The attack: Applying the Fiat-Shamir transformation using a concrete hash function h to the above sigma protocol yields the following non-interactive protocol:

1. **(Preprocessing):** The preprocessing algorithm receives a circuit C and outputs a digest of this circuit ψ . The honest prover is given C , and the verifier is given ψ .

^[11]This digest may contain a hash of C , and it allows the verifier to run in time sub-linear in the size of C .

2. **(Prover):** The honest prover generates a succinct commitment of the witness $m_1 = \text{Com}(\mathbf{w})$ and sends $(m_1, C, \mathbf{w})$ to the verifier.
3. **(Verifier):** The verifier computes $\beta = h(\psi, \mathbf{x}, \mathbf{y}, m_1)$, and accepts if and only if ψ is the digest of C , it holds that $m_1 = \text{Com}(\mathbf{w})$, and either $C(\mathbf{x}, \mathbf{w}) = \mathbf{y}$, or $C(\mathbf{x}, \mathbf{w}) = (0\|\beta)$.

Now, a malicious prover chooses a circuit $\tilde{C}$ and input/output pair $(\mathbf{x}, \mathbf{y})$ so that $\tilde{C}(\mathbf{x}, \mathbf{w}) \neq \mathbf{y}$ for every $\mathbf{w}$, and yet the prover causes the verifier to accept with probability 1:

- The input will be $\mathbf{x} = 0^n$ and the output will be $\mathbf{y} = 1^{r+1}$.
- The circuit $\tilde{C}(\mathbf{x}, \mathbf{w})$ is as follows: On input $\mathbf{x}$ and $\mathbf{w}$, parse $(\psi, \mathbf{y}) = \mathbf{w}$, and compute $m_1 = \text{Com}(\psi, \mathbf{y})$ and $\beta = h(\psi, \mathbf{x}, \mathbf{y}, m_1)$. Output $(0\|\beta)$.
- The malicious prover computes $m_1 = \text{Com}(\psi, \mathbf{y})$, and sends $(m_1, \tilde{C}, \mathbf{w} = (\psi, \mathbf{y}))$.

Since the output of $\tilde{C}$ always begins with 0, there will be no $\mathbf{w}$ so that $\tilde{C}(\mathbf{x}, \mathbf{w}) = \mathbf{y} = 1^{r+1}$, and so $(\tilde{C}, \mathbf{x}, \mathbf{y})$ ought to be rejected. However, observe that $\tilde{C}(\mathbf{x}, \mathbf{w}) = \tilde{C}(\mathbf{x}, (\psi, \mathbf{y})) = (0\|\beta)$, where $\beta = h(\psi, \mathbf{x}, \mathbf{y}, m_1)$ is the challenge that the verifier will compute. Therefore, the verifier will always accept.

The main crux behind this attack is that the maliciously generated circuit can correctly predict the verifier's second round message before it is actually generated. Such an attack falls under the family of *diagonalization attacks*. When the hash function h is replaced with a random oracle, this attack does not remain a concern for practical circuits C , which must be efficiently computable, and therefore cannot run a random oracle during its execution.

2.2 Our Solution: Use external randomness

Key Tool: Verifiable Random Function. An intuitive approach is to find a real-life instantiation that can computationally mimic the properties of a random oracle necessary for the Fiat-Shamir transformation. One such natural tool for this is a Verifiable Random Function (VRF). A Verifiable Random Function (VRF) is essentially a pseudorandom function (PRF) with an additional verification method to validate a PRF output. In particular, a VRF f has a key generation that generates a public key pk and a secret evaluation key sk [12], an evaluation function that takes as input x , and uses sk to output an evaluation y and a proof π , and a verification algorithm that uses the public key to check if (y, π) were indeed computed during the evaluation using input x . For technical reasons, we will use an enhancement of a VRF [13], called a simulatable VRF that has the following properties:

- An honestly generated output by the evaluation function should be correctly verified.
- There exists a simulator $(\text{SimGen}, \text{SimProve})$ such that for every input x (pk, y, π) generated by the real-world VRF algorithms is computationally indistinguishable from (pk', y', π') where $(\text{sk}', \text{pk}') \leftarrow \text{SimGen}(1^\lambda)$, y is uniformly sampled from an appropriate domain and $\pi' \leftarrow \text{SimProve}(\text{sk}', x, y)$.

For simplicity, here we imagine a setting where there is a *trusted third party* that is implementing a PRF service. As discussed prior, such a trust assumption is not necessary, but will make it simpler to explain the main contributions of our work. During the proof generation process, this service simply accepts queries from an appropriate domain and outputs the VRF evaluation at the queried point.

[12] There can also be an optional setup phase that outputs a Common Reference String (CRS)

[13] We want to argue pseudorandomness of the VRF output even when the corresponding proof is known. This property is ensured by a simulatable VRF.

Additionally, we also require our VRF to be stateful. Therefore, we introduce a novel variant of a stateful VRF, namely, a stateful and simulatable VRF (ssVRF). This is similar to a simulatable VRF with the ability to maintain state. In more detail, in an ssVRF,

- upon queried an input x , the algorithm first updates its current state from st' to st . Then, the evaluation function is fed the input (x, st) , and uses sk to output an evaluation y , state st and a proof π , and
- the verification algorithm that uses the public key to check if the triplet (y, st, π) is valid, i.e., whether (y, π) were indeed computed during the evaluation using input (x, st) .

A key property of a VRF (in the standard model) is that the verification *is a deterministic and local process that does not* need to make any evaluation queries. This property is crucial for one of our applications that we will describe later.

An ssVRF has all the properties of a simulatable VRF. Additionally, we also require the ssVRF to have “unforgeable” proofs. In other words, it should be hard for any polynomial sized adversary to produce a tuple (x, y, st, π) that gets verified without already querying x to the evaluation function *and* receiving (y, st, π) as the corresponding output. Please see Section 4 for a detailed and formal definition. We also discuss some constructions of ssVRF in Section 6 and note that several practical VRF constructions that are in fact deployed already satisfy this definition. Next, we present an outline of our construction.

The construction. The following is an outline of our relativized Fiat-Shamir transformation in the ssVRF oracle model applied to a Sigma protocol. Our construction is in the common reference string model.

Let $f := (f.Gen, f.E, f.V)$ be the ssVRF and let C be a circuit that takes as input an instance $\mathbf{x}$ and witness $\mathbf{w}$. The prover and the verifier are given oracle access to the evaluation function $f.E$. The prover wants to prove that $C(\mathbf{x}, \mathbf{w}) = 1$

- **Setup:** This generates the keys necessary for the ssVRF.
 - $(vk, sk) \leftarrow_{\S} f.Gen(1^\lambda, crs)$.
- **Preprocessing:** This takes as input the circuit C and honestly outputs a succinct digest ψ .
- **Honest Prover:**
 - Computes the first message α as produced by the interactive prover for the Sigma protocol. This is usually a commitment to $\mathbf{w}$.
 - Queries (ψ, x, α) to $f.E$ and receives (y, st, π) as output. (The state of $f.E$ gets internally updated upon execution.) y is interpreted as the verifier’s challenge.
 - Computes the final message γ using (α, y) as produced by the interactive prover for the Sigma protocol.
 - Outputs $(\alpha, (y, st, \pi), \gamma)$
- **Verifier:**
 - Check if $f.V(vk, ((\psi, x, \alpha), st), y, \pi) = 1$, else reject the proof.
 - Run the interactive verification protocol on (α, y, γ) , and return the result.

We refer to this as a non-interactive argument system relative to an ssVRF oracle.

2.3 Proof Outline

We prove straightline knowledge soundness of our construction via a reduction from the straightline round-by-round knowledge soundness of the Sigma protocol. Our proof follows that usual reduction technique where we use the malicious non-interactive prover Proof that breaks knowledge soundness of the non-interactive protocol after the Fiat-Shamir transformation to construct an interactive prover Proof^(I) that breaks the knowledge soundness of the interactive Sigma protocol. Since we will be in the relativized setting, all parties, including the interactive prover have oracle access to the ssVRF evaluation function $f.E$. The interactive prover Proof^(I) proceeds similar to the usual Fiat-Shamir reduction with a random oracle:

- It responds to every oracle query sent by Proof using the ssVRF simulator. Specifically, upon receiving a query of the form (ψ, x, α) , it first samples a random β , and then produces $(\pi, st) \leftarrow \text{SimProve}(\text{sk}, \beta, (\psi, x, \alpha), st')$, where sk is generated using the simulator's key generation process and st' is the current internal state that is maintained by Proof^(I).
- Assume that Proof makes q many oracle queries, then Proof^(I) picks one such query at random where it *programs* the verifier's challenge message β^* (which is a uniform random value). It again simulates the proof for β^* and sends it to Proof.
- After Proof terminates and returns a proof, Proof^(I) hopes that Proof indeed used the query where the interactive verifier's challenge was program. If so, then it outputs the final message.

The proof follows via a hybrid argument. At a high level,

1. First, we switch the true ssVRF in the interactive protocol with the VRF simulator. This switch is computationally indistinguishable because of the *simulatability* of the VRF.
2. At this point, Proof^(I) simply forwards the queries made by Proof to this external simulator, rather than producing the simulated proofs locally. It continues to program and simulate the proof locally for the query where the challenge is planted. This change is undetectable to Proof, and the view of the interactive verifier and preprocessing algorithm do not change.
3. At this step, we will move to a hybrid where the challenge is no longer programmed, but is sampled uniformly and independently of the interactive verifier's challenge. Indeed, this switch is undetectable to the non-interactive prover Proof because β^* was also picked uniformly at random to begin with. However, we need to ensure the behavior of the verifier does not change when the proof is generated with respect to the programmed vs. non-programmed simulator. This is not an issue in the standard Fiat-Shamir because the verifier do not have access to this oracle. However, in the relativized world, we have to ensure that neither the verifier can query this programmed location and detect the change. One way for the verifier detect the change is if (1) they query (ψ^*, x^*, α^*) to $f.E$ and receive (β^*, st^*, π^*) as the output. This cannot happen due to the uniqueness of the state, and (2) if the previous event does not occur, but the verifier can forge (β^*, st^*, π^*) , and run the ssVRF verification on $(\psi^*, x^*, \alpha^*, \beta^*, st^*, \pi^*)$. This contradicts the proof unforgeability of ssVRF. Secondly the verifier can also detect the change if it runs the ssVRF on the FS challenge as input. This is avoided by padding a string of all 0s that is longer than the length of the proof verification circuit to the input that is used to compute the FS challenge. Usually the verifier is a small program so this does not have significant impact on efficiency.
4. Finally, we will again use the simulatability to switch the ssVRF simulation to real evaluations.

The formal proof is slightly more nuanced. We refer to Section 5 for details.

Remark 2.1. Our construction in the relativized model does not give VRF oracle access to the circuit C which was the case in [AY25]. Typically, a prover publishes a proof attesting to the validity of an NP statement, i.e., it claims that $\exists w : C(x, w) = 1$ which is well defined. When the circuit is given oracle access to a *deterministic function*, say a random oracle, we move to relativized NP where every choice of the random oracle defines a unique language. However, once the oracle is fixed, the language is well-defined and this model has also been widely studied in the literature [BCG24]. On the other hand, in our case since $f.E$ adds a new state to every query, it is possible that $C^{f.E}(x, w)$ does not have the same outcome when the circuit is evaluated multiple times. Specifically, it is possible that the apparent membership of (C, x) in the NP language changes with every invocation of C . It is unclear how to model this meaningfully in the definition of a proof system.

Having said that, we stress that our model captures a very practical scenario. The statement circuit is non-relativized because real proof statements are ordinary deterministic computations. The prover, however, is an external malicious algorithm and can query the VRF service during proof generation. Our model rule out the practical diagonalization attack that applies to public Fiat–Shamir hashes, because the circuit cannot compute the secret-keyed external VRF evaluation used to generate the challenge.

2.4 Extending to non-relativized model and an application to recursive proofs:

Our result carries forward and has meaningful implications to the non-relativized world as well, i.e., the interactive prover and verifier do not get the ssVRF, and it is only used in the Fiat-Shamir transformation to generate the verifier’s challenge message, which is akin to the standard Fiat-Shamir. We provide the formal construction and a proof outline in Section 7 for completeness, but it follows in a straightforward manner from the relativized result. In fact, we can show that in the absence of oracle access to the circuit, using a plain simulatable oracle without additional state suffices and our result extends to all circuits in P/poly .^[14] Along with providing a practically feasible and provably secure instantiation of the Fiat-Shamir transformation, this result also has interesting implications for recursive proofs.

Recursive Succinct Proofs. Succinct proofs or arguments enable a prover to convince a verifier that a computation was executed correctly, with proof sizes and verification times asymptotically smaller than the computation itself. Essentially, they allow succinctness in a single proof instance. However, many applications demand the ability to verify and aggregate multiple proofs or computations into a single succinct proof. Such a requirement comes up naturally in several real-world applications such as, Zero-knowledge Rollup [CLKL22], Incremental Verifiable Computation (IVC) [Val08], Proof aggregation and compression [GMN22, BCMS20b, BCCT13], and many more.

The requirement in all the above applications is to produce a *proof about the knowledge of another proof*, rather than proving something about the statement directly. In other words, rather than generating a proof π that “we know a witness w for a statement x ”, we need to generate a proof π' to attest to the fact that “we know a proof π that attests to a witness w for the statement x .” As an example, consider the following two-step recursion:

Let $(\text{Setup}_1, \text{Pre}_1, P_1, V_1)$ be a (pre-processing) succinct proof and let $\mathcal{L} = \{(C, x) : \exists w \text{ such that } C(x, w) = 1\}$ be an NP statement. In a recursive proof, the prover shall generate a proof for the following statement:

$$\mathcal{L}' = \{(C, x, \text{crs}) : \exists \pi \text{ such that } V_1(\text{Pre}(\text{crs}, C), x, \pi) = 1\}, \text{ where} \\ \pi \leftarrow \text{Proof}_1(\text{crs}, C, \psi, x, w), \text{ and } \text{crs} \leftarrow \text{Setup}_1(1^\lambda).$$

^[14]We note that, restricting circuits not to have oracle access also eliminates the possibility of any diagonalization attack for FS or XFS. However, their security proofs do not work without random oracles. In contrast, our construction continues to have provable security in the standard model if a standard model VRF is used.

Observe that the statement of interest in the second level of the recursive proof is about the *verification algorithm* of the first level proof. This poses a major challenge when constructing recursive proofs, even in an idealized model. In particular, if we use a non-interactive proof in the random oracle model, then the recursion breaks because the verifier V_1 defined above also has to access the random oracle. Therefore the language $\mathcal{L}'$ actually becomes oracle dependent, and therefore a relativized language. As discussed before, there are several known barriers for constructing provably secure succinct proofs for circuits relative to a random oracle [Val08, HAN23, BCG24]. Thus, the standard practice is to instantiate the random oracle with a hash function and hope that the heuristic is secure. Therefore, as the state of affairs stand today, we can either only provide heuristic guarantees [Val08] or rely on non-standard knowledge based or idealized model assumptions [CCG⁺23, BCCT13, BCTV14, KST22, BCMS20b, BCL⁺21, BDFG21], or use heavy tools like iO [DJJ⁺25] to construct recursive proofs for NP.

If $(\text{Setup}_1, \text{Pre}_1, P_1, V_1)$ is instantiated using our ssVRF based construction, then V_1 *does not* need oracle access as the VRF verification is a local operation. This, in turn makes $\mathcal{L}'$ a *non-relativized* language, where we can again use the succinct proof recursively. Thus, as a corollary to our non-relativized Fiat-Shamir in the simulatable VRF model, we achieve provably secure recursive proofs. This is elaborated in Section 7.1 and a concrete instantiation with NOVA is provided in Section 7.2.

What happens if we use a PRF? A PRF is the most natural real world instantiation of a random oracle. So, it is natural to ask if we could have simplified our construction by simply using a PRF oracle. The hope will be that a PRF can emulate the properties of a random oracle and, at the same time, evade diagonalization attacks because the circuit cannot compute the FS challenge without knowing the PRF evaluation key. While it is plausible that a PRF will suffice to instantiate provably secure FS transformation, the main drawback is that a PRF does not support any procedure for local verification. This means that the only way for the verifier to check if the FS challenge is honestly computed is to query the PRF service provider on the appropriate input and check if the output indeed matches the challenge. While it might work well for the base proof, such an FS transformation will not yield provable security during recursive compositions. Indeed, in the recursive step, the circuit for which the proof needs to be generated will comprise of the base verifier circuit, and thus will need oracle access to the PRF. Such a circuit cannot even be described efficiently, and therefore it is unclear how to prove soundness in such a setting without reverting back to heuristics used in the random oracle model. Moreover, using a PRF can also cause efficiency bottlenecks in practice. In numerous real-world application, proof generation happens once but verification is performed multiple times but several entities across varying time period. Using a PRF will force every verification to make a separate PRF query, thereby, creating a significant bottleneck.

3 Preliminaries

3.1 Relativized Sigma Proof with Pre-Processing

We first describe the notion of a pre-processing sigma protocol for boolean circuits relative to a function f from a family $\mathcal{F}$. This notion was defined in [AY25] and we use a slight variation of their model. Essentially, our model is similar to a standard sigma protocol, except the prover and verifier have oracle access to a function f , while the circuit does not^[15]. Our definition additionally includes a setup phase that is completely independent of the circuit or instance used in the proof. The purpose of the setup is to select a function $f \in \mathcal{F}$.

Remark 3.1. Typically, one does not assume a separate setup phase to sample the oracle. However, putting the definition in this form will make it easier for instantiations later on.

^[15]We elaborate the reasoning behind this choice in Remark 2.1

Definition 3.2 (Relativized pre-processing Sigma Proof with Setup). Let $\mathcal{F} : \{0, 1\}^n \rightarrow \{0, 1\}^m$ be any function family. A pre-processing Sigma protocol with respect to $\mathcal{F}$ is a tuple of five algorithms (Setup, Pre, Proof₁, Proof₂, Verify) such that:

- $f \leftarrow \text{Setup}(1^\lambda)$: is an (inefficient) algorithm that takes as input the security parameter and selects a function $f \in \mathcal{F}$ that is given as oracle access to the other algorithms.
- $\psi \leftarrow \text{Pre}^{f(\cdot)}(C, 1^\lambda)$ is a PPT algorithm with oracle access to a function $f \in \mathcal{F}$ that takes as input the circuit C , the security parameter λ , and outputs a digest ψ .
- $(\alpha, \text{st}) \leftarrow \text{Proof}_1^{f(\cdot)}(C, \psi, x, w)$ is a PPT algorithm with oracle access to a function $f \in \mathcal{F}$ and takes as input the circuit C , digest ψ , input x and witness w , and outputs its first message α and state st .
- $\gamma \leftarrow \text{Proof}_2^{f(\cdot)}(C, \psi, x, w, \alpha, \beta)$ is a PPT algorithm with oracle access to $f \in \mathcal{F}$ that takes as input the circuit C , digest ψ , input x and witness w , and the first two round messages α, β , and outputs its final round message γ .
- $0/1 \leftarrow \text{Verify}^{f(\cdot)}(\psi, x, \alpha, \beta, \gamma)$ is a deterministic algorithm that has oracle access to $f \in \mathcal{F}$ and takes as input the digest ψ , input x and proof transcript α, β, γ , and either accepts or rejects the proof.

The above tuple of algorithms give rise to a honest verifier sigma protocol for a family of circuits $\mathcal{C}$ if the following hold:

- **Completeness.** For every circuit $C \in \mathcal{C}$, function $f \in \mathcal{F}$, inputs (x, w) such that $C(x, w) = 1$ it holds that

$$\Pr \left[\text{Verify}^{f(\cdot)}(\psi, x, \alpha, \beta, \gamma) = 1 \mid \begin{array}{l} \psi \leftarrow \text{Pre}^{f(\cdot)}(C, 1^\lambda) \\ (\alpha, \text{st}) \leftarrow \text{Proof}_1^{f(\cdot)}(C, \psi, x, w) \\ \beta \leftarrow_{\$} \{0, 1\}^m \\ \gamma \leftarrow \text{Proof}_2^{f(\cdot)}(C, \psi, x, w, \alpha, \beta, \text{st}) \end{array} \right] = 1.$$

- **Adaptive (straightline) round-by-round knowledge soundness.** Let $\lambda \in \mathbb{N}$ be the security parameter. There exists a polynomial-time extractor E such that the following holds:

1. For all malicious PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ making at most polynomial many oracle queries, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[\begin{array}{l} C \in \mathcal{C}, \\ \wedge \text{Verify}^{f(\cdot)}(\psi, x, \alpha, \beta, \gamma) = 1, \\ \wedge C(x, w) = 0. \end{array} \mid \begin{array}{l} f \leftarrow_{\$} \text{Setup}(1^\lambda), \\ (C, \text{state}_1) \leftarrow \mathcal{A}_1^{f(\cdot)}(1^\lambda), \\ \psi \leftarrow \text{Pre}^{f(\cdot)}(C), \\ (x, \alpha, \text{state}_2) \leftarrow \mathcal{A}_2^{f(\cdot)}(C, \text{state}_1), \\ \text{tr} := \text{all queries made to the oracle so far}, \\ w \leftarrow E(C, x, \alpha, \text{state}_2, \text{tr}), \\ \beta \leftarrow_{\$} \{0, 1\}^m, \\ \gamma \leftarrow \mathcal{A}_3^{f(\cdot)}(C, x, \alpha, \beta, \text{state}_2). \end{array} \right] \leq \text{negl}(\lambda).$$

2. For all malicious PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ making at most polynomial many oracle queries, and every PPT adversary $\mathcal{B}$ making at most polynomial many oracle queries there

exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[\begin{array}{l} C \in \mathcal{C}, \\ \wedge w \neq w'. \end{array} \left| \begin{array}{l} f \leftarrow_{\$} \text{Setup}(1^\lambda), \\ (C, \text{state}_1) \leftarrow \mathcal{A}_1^{f(\cdot)}(1^\lambda), \\ \psi \leftarrow \text{Pre}^{f(\cdot)}(C), \\ (x, \alpha, \text{state}_2) \leftarrow \mathcal{A}_2^{f(\cdot)}(C, \text{state}_1), \\ \text{tr} := \text{all queries made to the oracle so far}, \\ w \leftarrow E(C, x, \alpha, \text{state}_2, \text{tr}), \\ \beta \leftarrow_{\$} \{0, 1\}^m, \\ \perp \leftarrow \mathcal{B}^{f(\cdot)}(\beta, \text{state}_2), \\ \text{tr}' := \text{all queries made to the oracle by } \mathcal{B}, \\ w' \leftarrow E(C, x, \alpha, \text{state}_2, \text{tr} \cup \text{tr}'). \end{array} \right. \right] \leq \text{negl}(\lambda).$$

Definition 3.3 (Relativized Non-Interactive Argument System). A quadruplet of algorithms (Setup, Pre, Proof, Verify) is a relativized non-interactive argument with respect to oracles $\mathcal{F}$ for a family of Boolean circuits $\mathcal{C}$ if the following properties hold:

- **Completeness.** For every $C \in \mathcal{C}$, x, w , and oracle $f \in \mathcal{F}$, for which $C(x, w) = 1$, it holds that

$$\Pr \left[\text{Verify}^{f(\cdot)}(\psi, x, \pi) = 1 \left| \begin{array}{l} \pi \leftarrow \text{Proof}^{f(\cdot)}(C, x, w) \\ \psi \leftarrow \text{Pre}^{f(\cdot)}(C) \end{array} \right. \right] = 1.$$

- **Adaptive soundness.** For every malicious t -query adversary $\mathcal{A}$, there exists a negligible function $\varepsilon(\cdot)$ such that

$$\Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \forall w, C(x, w) = 0 \\ \wedge \text{Verify}^{f(\cdot)}(\psi, x, \pi) = 1 \end{array} \left| \begin{array}{l} f \leftarrow_{\$} \mathcal{F} \\ (C, x, \pi) \leftarrow \mathcal{A}^{f(\cdot)}(1^\lambda) \\ \psi \leftarrow \text{Pre}^{f(\cdot)}(C) \end{array} \right. \right] \leq \varepsilon(\lambda, t).$$

- **Adaptive (straightline) knowledge soundness.** Let $\lambda \in \mathbb{N}$ be the security parameter. There exists a polynomial-time extractor E such that for all malicious PPT adversary $\mathcal{A}$ making at most polynomial many oracle queries, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge C(x, w) = 0 \\ \wedge \text{Verify}^{f(\cdot)}(\psi, x, \pi) = 1 \end{array} \left| \begin{array}{l} f \leftarrow_{\$} \text{Setup}(1^\lambda) \\ (C, x, \pi) \leftarrow \mathcal{A}^{f(\cdot)}(1^\lambda) \\ \text{tr} := \text{all queries made to the oracle so far} \\ w \leftarrow E(C, x, \pi, \text{tr}) \\ \psi \leftarrow \text{Pre}^{f(\cdot)}(C) \end{array} \right. \right] \leq \varepsilon(\lambda, t).$$

Relativized proofs in the Random Oracle Model [AY25]. A (pre-processing sigma or non-interactive) proof is said to be relativized in the random oracle model if $\mathcal{F} : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is a random oracle from n to m bits.

3.2 Simulatable Verifiable Random Function [CL07]

Definition 3.4 (sVRF). A simulatable VRF (sVRF) is defined by a tuple of four PPT algorithms, namely $f := (f.\text{Setup}, f.\text{Gen}, f.E, f.V)$ with syntax:

- $f.\text{Setup}(1^\lambda) \rightarrow (\text{crs}, \text{td})$: The (optional) randomized setup algorithm takes as input the security parameter λ and outputs a public setup string crs and a trapdoor td . The crs is used in the other three algorithms as input.
 - $f.\text{Gen}(1^\lambda, \text{crs}) \rightarrow (\text{vk}, \text{sk})$: A randomized key-generation algorithm that takes as input the security parameter λ and the crs , and outputs a key pair (vk, sk) , where vk is the public verification key and sk is the secret key.
 - $f.\text{E}(\text{sk}, x, \text{crs}) \rightarrow (y, \pi)$: A deterministic evaluation algorithm that takes as input a secret key sk , input x , and crs , and outputs y and a proof π .
 - $f.\text{V}(\text{vk}, x, y, \pi, \text{crs}) \rightarrow b \in \{0, 1\}$: A deterministic algorithm that takes as input the verification key vk , input x , output y and proof π , and outputs a bit denoting whether the proof verified or not.
1. **Correctness:** For all $\lambda \in \mathbb{N}$, let $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$, $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$. Then, for any x , if $(y, \pi) \leftarrow f.\text{E}(\text{sk}, x, \text{crs})$, then it holds that:

$$f.\text{V}(\text{vk}, x, y, \pi, \text{crs}) = 1.$$

2. **Output Uniqueness:** For all $\lambda \in \mathbb{N}$, let $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$. Then, for any x there does not exist any $(y_0, \pi_0) \neq (y_1, \pi_1)$ such that $\text{Verify}(\text{vk}, x, y_0, \pi_0, \text{crs}) = \text{Verify}(\text{vk}, x, y_1, \pi_1, \text{crs}) = 1$.
3. **Simulatability:** For any $\lambda \in \mathbb{N}$, there exists a simulator, which is a tuple of algorithms $(\text{SimSetup}, \text{SimGen}, \text{SimProve})$ such that for all PPT adversary $\mathcal{A}$, $\mathcal{A}$'s views in the following two games are indistinguishable:
 - **Game Real.**
 - $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$; return crs to $\mathcal{A}$;
 - On key generation query from $\mathcal{A}$, run $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$ and return vk to $\mathcal{A}$; store (vk, sk) .
 - On query (vk, x) check if there exists a (vk, sk) in the memory, if not return “error”; otherwise return $(y, \pi) \leftarrow f.\text{E}(\text{sk}, x, \text{crs})$.
 - **Game Ideal.**
 - $(\text{crs}, \text{td}) \leftarrow \text{SimSetup}(1^\lambda)$; return crs to $\mathcal{A}$;
 - On key generation query from $\mathcal{A}$, run $(\text{vk}, \text{sk}) \leftarrow \text{SimGen}(1^\lambda, \text{td}, \text{crs})$ and return vk to $\mathcal{A}$; store (vk, sk) .
 - On query (vk, x) check if there exists a (vk, sk) in the memory, if not return “error”; otherwise return (y, π) such that y is chosen randomly from its domain, and $\pi \leftarrow \text{SimProve}(\text{sk}, \text{td}, y, x, \text{crs})$. Here, SimProve is also a stateful algorithm.
4. **Unforgeability:** For all $\lambda \in \mathbb{N}$, and for all polynomial sized admissible adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[f.\text{V}(\text{vk}, x, y, \pi, \text{crs}) = 1 \mid \begin{array}{l} \text{crs} \leftarrow f.\text{Setup}(1^\lambda); \\ (\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs}); \\ (x, y, \pi) \leftarrow \mathcal{A}^{f.\text{E}(\text{sk}, \cdot, \text{crs})}(\text{crs}, \text{vk}) \end{array} \right] \leq \text{negl}(\lambda).$$

Let Tr denote the list of all input-output pairs generated during the oracle queries to $f.\text{E}$ by $\mathcal{A}$. $\mathcal{A}$ is admissible if $(x, y, \pi) \notin \text{Tr}$.

For an sVRF, unforgeability follows from uniqueness and simulatability. We state it separately because it will be useful for the proofs.

4 Stateful Simulatable VRFs

We start with the definition of our new notion of a stateful simulatable VRF which is an enhancement of the standard notion of simulatable VRFs with some additional properties. In fact, our definition captures the natural functionalities offered by blockchain-based VRF services, such as Chainlink [Lab22], Binance [Ora23], Supra [Res23], Harmony [Har23] etc., as we will describe in section 6.

We work with some arbitrary domains for each variable, such as key, input, output, and proof, without specifying them explicitly below. One can pick appropriate domains based on specific instantiations.

Definition 4.1 (ssVRF). *A stateful and simulatable VRF (ssVRF) is defined by a tuple of four PPT algorithms, namely $f := (f.\text{Setup}, f.\text{Gen}, f.\text{E}, f.\text{V})$ with syntax:*

- $f.\text{Setup}(1^\lambda) \rightarrow \text{crs}$: *The (optional) randomized setup algorithm takes as input the security parameter λ and outputs a public setup string crs . The crs is used in the other three algorithms as input.*
- $f.\text{Gen}(1^\lambda, \text{crs}) \rightarrow (\text{vk}, \text{sk})$: *A randomized key-generation algorithm that takes as input the security parameter λ and the crs , and outputs a key pair (vk, sk) , where vk is the public verification key and sk is the secret key.*
- $f.\text{E}(\text{sk}, x, \text{st}, \text{crs}) \rightarrow (y, \pi, \text{st}')$: *A deterministic evaluation algorithm that takes as input a secret key sk and input x , crs and an internal state st , and outputs y and a proof π along with the updated state st' .*

This state st' will be used by $f.\text{E}$ as its internal state on the next execution.

- $f.\text{V}(\text{vk}, (x, \text{st}), y, \pi, \text{crs}) \rightarrow b \in \{0, 1\}$: *A deterministic algorithm that takes as input the verification key vk , input x , output y , proof π and a state st , and outputs a bit denoting whether the proof verified or not.*
1. **Correctness:** *For all $\lambda \in \mathbb{N}$, let $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$, $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$. Then, for any (x, st) , if $(y, \pi, \text{st}') \leftarrow f.\text{E}(\text{sk}, (x, \text{st}), \text{crs})$, then it holds that:*

$$f.\text{V}(\text{vk}, (x, \text{st}'), y, \pi, \text{crs}) = 1.$$

2. **Output Uniqueness:** *For all $\lambda \in \mathbb{N}$, let $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$. Then, for any (x, st) there does not exist any $(y_0, \pi_0) \neq (y_1, \pi_1)$ such that $\text{Verify}(\text{vk}, (x, \text{st}), y_0, \pi_0, \text{crs}) = \text{Verify}(\text{vk}, (x, \text{st}), y_1, \pi_1, \text{crs}) = 1$.*
3. **State Uniqueness:** *For all $\lambda \in \mathbb{N}$ and any $t = \text{poly}(\lambda) \in \mathbb{N}$ and any set of $\{x_i\}_{i \in [t]}$, there exists a negligible function $\text{negl}(\cdot)$ such that the following probability is at most $\text{negl}(\lambda)$:*

$$\Pr \left[\exists i, j \in [t] : (\text{st}_i = \text{st}_j) \wedge (i \neq j) \mid \begin{array}{l} \text{crs} \leftarrow f.\text{Setup}(1^\lambda); \\ (\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs}); \\ \text{st}_0 := \perp; \\ \forall i \in [t] : (y_i, \pi_i, \text{st}_i) \leftarrow f.\text{E}(\text{sk}, x_i, \text{st}_{i-1}, \text{crs}) \end{array} \right]$$

4. **Unforgeability:** *For all $\lambda \in \mathbb{N}$, and for all polynomial sized admissible adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that*

$$\Pr \left[f.\text{V}(\text{vk}, (x, \text{st}), y, \pi, \text{crs}) = 1 \mid \begin{array}{l} \text{crs} \leftarrow f.\text{Setup}(1^\lambda); \\ (\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs}); \\ (x, y, \text{st}, \pi) \leftarrow \mathcal{A}^{f.\text{E}(\text{sk}, \cdot, \text{crs})}(\text{crs}, \text{vk}) \end{array} \right] \leq \text{negl}(\lambda).$$

Let Tr denote the list of all input-output pairs generated during the oracle queries to $f.\text{E}$ by $\mathcal{A}$. $\mathcal{A}$ is admissible if $(x, \text{st}, y, \pi) \notin \text{Tr}$.

5. **Simulatability:** For any $\lambda \in \mathbb{N}$, there exists a simulator, which is a tuple of algorithms $(\text{SimSetup}, \text{SimGen}, \text{SimProve})$ such that for all PPT adversary $\mathcal{A}$, $\mathcal{A}$'s views in the following two games are indistinguishable:

- **Game Real.**

- $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$; return crs to $\mathcal{A}$;
- On key generation query from $\mathcal{A}$, run $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$ and return vk to $\mathcal{A}$; initialize the state $\text{st} := \perp$ and store (vk, sk) .
- On query (vk, x) check if there exists a (vk, sk) in the memory, if not return “error”; otherwise return $(y, \text{st}', \pi) \leftarrow f.E(\text{sk}, (x, \text{st}), \text{crs})$.

- **Game Ideal.**

- $(\text{crs}, \text{td}) \leftarrow \text{SimSetup}(1^\lambda)$; return crs to $\mathcal{A}$;
- On key generation query from $\mathcal{A}$, run $(\text{vk}, \text{sk}) \leftarrow \text{SimGen}(1^\lambda, \text{td}, \text{crs})$ and return vk to $\mathcal{A}$; initialize the state $\text{st} := \perp$ and store (vk, sk) .
- On query (vk, x) check if there exists a (vk, sk) in the memory, if not return “error”; otherwise return (y, st', π) such that y is chosen randomly from its domain, and $(\text{st}', \pi) \leftarrow \text{SimProve}(\text{sk}, \text{td}, y, x, \text{st}, \text{crs})$. Here, SimProve is also a stateful algorithm.

State uniqueness and unforgeability in the Ideal world. Observe that if the state uniqueness and unforgeability hold when the evaluations are generated in the real world using $(f.\text{Setup}, f.\text{Gen}, f.E)$, then simulatability implies that they must also hold in the ideal world using $(\text{SimSetup}, \text{SimGen}, \text{SimProve})$. If state uniqueness does not hold in the ideal world, then an adversary can query the evaluation function polynomially many times and check if there is a collision in the states output or not to check whether it is interacting with the real game or the ideal game, and therefore break simulatability. Similarly any adversary that can forge a proof in the ideal world and cannot forge in the real world can also be a distinguisher for the simulatability game.

5 Fiat-Shamir in the relativized VRF oracle model

VRF oracle model:. We introduce the VRF oracle model, which is similar to the usual Fiat-Shamir in the random oracle model, except that the verifier's challenge is sampled via oracle access to a Verifiable Random Function with appropriate properties, as opposed to a random oracle. In this model, a setup algorithm generates the evaluation key and the public key for the VRF. A *trusted third party* is given access to the secret evaluation key that it uses to output VRF evaluations and proofs. Anybody with access to the VRF public key can verify the VRF proofs. Intuitively, in this model, rather than querying the random oracle to check if the prover generated the challenge honestly, the verifier runs the VRF verification locally.

This can be naturally extended to the relativized VRF oracle model where the interactive prover and verifier are allowed to have oracle access to the VRF. We first show that the Fiat-Shamir transformation is provably secure in the relativized VRF oracle model.

5.1 Fiat-Shamir transformation in the relativized VRF model

For notational simplicity we use $f.E_{\text{sk}}$ to denote the ssVRF evaluation oracle hardcoded with sk as the secret key. Let v denote an upper bound on the length of the description of $\text{Verify}^{(I)}(\cdot)$. 0^v denotes a string of all 0s of length v .

Construction 1 (Non-interactive argument relative to ssVRF).

- $\text{Setup}(1^\lambda)$:
 - $\text{crs} \leftarrow_{\$} f.\text{Setup}(1^\lambda)$.
 - $(vk, sk) \leftarrow_{\$} f.\text{Gen}(1^\lambda, \text{crs})$.
 - $h \leftarrow_{\$} \text{Hash.Gen}(1^\lambda)$.
 - Output (crs, vk, h) .
- $\text{Pre}^{f.E_{sk}}(\text{crs}, vk, h, C)$:
 - $\psi \leftarrow_{\$} \text{Pre}^{(I)(f.E_{sk})}(h, C)$.
- $\text{Proof}^{f.E_{sk}}(\text{crs}, vk, h, C, \psi, x, w)$:
 - $\alpha \leftarrow_{\$} \text{Proof}_1^{(I)(f.E_{sk})}(C, \psi, x, w)$.
 - $(y, \text{st}', \pi) \leftarrow f.E_{sk}((0^v, \psi, x, \alpha), \text{st}, \text{crs})$.
 - $\beta := y$.
 - $\gamma \leftarrow_{\$} \text{Proof}_2^{(I)(f.E_{sk})}(C, \psi, x, w, \alpha, \beta)$.
 - $\Pi := (\alpha, \beta, \gamma, \text{st}', \pi)$.
- $\text{Verify}^{f.E_{sk}}(\text{crs}, vk, h, \psi, x, \Pi)$:
 - Parse Π as $(\alpha, \beta, \gamma, \text{st}, \pi)$.
 - If $f.V(vk, ((0^v, \psi, x, \alpha), \text{st}), \beta, \pi, \text{crs}) \neq 1$, output 0.
 - Else output $\text{Verify}^{(I)(f.E_{sk})}(\psi, x, (\alpha, \beta, \gamma))$.

Remark 5.1. In several cases, the challenge space is smaller than that of the first round commitment sent by the prover. On the other hand, the VRF evaluation will expand the input length. To address this, one can simply truncate the VRF evaluation to appropriate length and append the remaining VRF output bits with the proof. Another option can be to use a collision resistant hash function on the input to the VRF to compress it to an arbitrary length before evaluating. We skip this detail for ease of exposition.

Theorem 5.2. *Let $\lambda \in \mathbb{N}$ be the security parameter. Assume*

- $(f.\text{Setup}, f.\text{Gen}, f.E, f.V)$ is an ssVRF satisfying all properties of Definition 4.1, and
- $(\text{Pre}^{(I)}, \text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Verify}^{(I)})$ is a sigma protocol relative to f that satisfies adaptive round-by-round knowledge soundness and every algorithm can make at most $q = \text{poly}(\lambda)$ many VRF oracle queries.

Then, construction 1 is a non-interactive argument system with adaptive knowledge soundness relative to f for every circuit C .

Proof. Completeness. This is immediate from the completeness of the interactive proof and verification correction of ssVRF.

Knowledge Soundness.

We will prove knowledge soundness of the relativized non-interactive proof via reduction from the knowledge soundness of the relativized interactive proof. In order to do so, we will first prove the following theorem.

Theorem 5.3. Let $\lambda \in \mathbb{N}$ be the security parameter. For every extractor $E^{(I)}$, there exists an interactive prover $\text{Proof}^{(I)} = (\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)})$ and an extractor $\mathcal{E}$ for the non-interactive proof $(\text{Setup}, \text{Pre}, \text{Proof} = (\text{Proof}_1, \text{Proof}_2, \text{Proof}_3), \text{Verify})$ such that for all polynomial sized adversaries $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ making at most q many oracle queries, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.E_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \pi)) = 1 \\ \wedge C(x, w) = 0 \end{array} \right] \left[\begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, \text{st}_1) \leftarrow \mathcal{B}_1^{(f.E_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(f.E_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \pi, \text{st}_2) \leftarrow \mathcal{B}_2^{(f.E_{\text{sk}})}(\text{vk}, \text{crs}, h, C, \text{st}_1), \\ \text{tr} := \text{all queries made to the oracle}, \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_2, \text{tr}) \end{array} \right] \\ \leq q \cdot \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(I)(f.E_{\text{sk}})}(\psi, x, \alpha, \beta, \gamma) = 1 \\ \wedge C(\hat{x}, w) = 0 \end{array} \right] \left[\begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, \text{state}_1) \leftarrow \text{Proof}_1^{(f.E_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(I)(f.E_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \text{state}_2) \leftarrow \text{Proof}_2^{(I)(f.E_{\text{sk}})}(\text{vk}, \text{crs}, h, C, \text{state}_1), \\ \text{tr} := \text{all queries made to the oracle so far}, \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{state}_2, \text{tr}), \\ \hat{\beta} \leftarrow_{\$} \{0, 1\}^m \\ \hat{\gamma} \leftarrow \text{Proof}_3^{(I)(f.E_{\text{sk}})}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \text{state}_2). \end{array} \right] + \text{negl}(\lambda)$$

Proof. Let $E^{(I)}$ be the extractor for the interactive proof system. We define $\mathcal{E}$ as follows:

$\mathcal{E}(C, x, \alpha, \beta, \gamma, \text{st}, \pi, \text{st}_2, \text{tr})$:

- Output $w \leftarrow E^{(I)}(C, x, \alpha, \text{st}_2, \text{tr})$.

Similarly, we define $\text{Proof}^{(I)} := (\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)})$ as follows:

$\text{Proof}^{(I)}(\text{crs}, \text{vk}, h)$:

- $(\tilde{\text{crs}}, \tilde{\text{td}}) \leftarrow \text{SimSetup}(1^\lambda)$.
- $(\tilde{\text{vk}}, \tilde{\text{sk}}) \leftarrow \text{SimGen}(1^\lambda, \tilde{\text{td}}, \tilde{\text{crs}})$.

Define the following oracle as E' that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms $(\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)})$ can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$
- For i^{th} query q_i made by $\mathcal{B}$ to $f.E_{\text{sk}}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $\tilde{\beta}_i \leftarrow_{\$} \{0, 1\}^m$.

2. $(\tilde{\text{st}}_i, \tilde{\pi}_i) \leftarrow \text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \beta_i, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 3. Store $(\psi_i, x_i, \alpha_i, \tilde{\text{st}}_i, \tilde{\pi}_i, \tilde{\beta}_i)$ in tr .
 4. Return $(\tilde{\text{st}}_i, \tilde{\pi}_i, \tilde{\beta}_i)$.
- * If $i = i^*$,
1. Send $(\psi_{i^*}, x_{i^*}, \alpha_{i^*})$ as the first round message for the interactive proof.
 2. Let β_{i^*} be the challenge received.
 3. $(\tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}) \leftarrow \text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \beta^*, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 4. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}, \beta_{i^*})$ to tr .
 5. Let tr^* denotes all oracle queries made till the i^* th query,
 6. Let $state_2$ be the auxilliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 7. Return $(\tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I)(f.E_{\text{sk}})}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, st_1) \leftarrow \mathcal{B}_1^{(E')}(c\tilde{\text{rs}}, \tilde{\text{vk}}, h)$.
 3. $state_1 := (st_1, tr)$
 4. Return C to the interactive verifier.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I)(f.E_{\text{sk}})}(\text{crs}, \text{vk}, C, state_1)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{E'}(1^\lambda, c\tilde{\text{rs}}, \tilde{\text{vk}}, C, st_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \tilde{\text{st}}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.

Remark 5.4. Note that $\text{Proof}'^{(I)}$ can be appropriately split into $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$ before and after the i^* -th query. We do not do so explicitly in the algorithm above for the sake of presentation.

We will first define an event called Good where if $(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})$ is the output by $\mathcal{B}_2$ as the challenge then

Good :

(Let $\text{Pre}^{(f.E_{\text{sk}})}(C) = \psi) \wedge (\psi, \hat{x}, \hat{\alpha})$ was used by $\mathcal{B}$ as an oracle query to $f.E$ and the response was $(\hat{\text{st}}, \hat{\beta}, \hat{\pi})$
 $\wedge E^{(I)}(C, \hat{x}, \hat{\alpha}, st_2, tr) = E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$).

By the law of total probability,

$$\begin{aligned}
& \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.\text{E}_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \pi)) = 1 \\ \wedge C(\hat{x}, w) = 0 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{(f.\text{E}_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(f.\text{E}_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \pi, st_2) \leftarrow \mathcal{B}_2^{(f.\text{E}_{\text{sk}})}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&= \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.\text{E}_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{(f.\text{E}_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(f.\text{E}_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{(f.\text{E}_{\text{sk}})}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&+ \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.\text{E}_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{(f.\text{E}_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(f.\text{E}_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{(f.\text{E}_{\text{sk}})}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right]
\end{aligned}$$

We will prove the theorem in two parts.

I. Show that

$$\begin{aligned}
& \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.\text{E}_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{(f.\text{E}_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(f.\text{E}_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{(f.\text{E}_{\text{sk}})}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&\leq q \cdot \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(I)(f.\text{E}_{\text{sk}})}(\psi, \hat{x}, \hat{\alpha}, \beta, \hat{\gamma}) = 1 \\ \wedge C(x, w) = 0 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, state_1) \leftarrow \text{Proof}_1^{(f.\text{E}_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(I)(f.\text{E}_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, state_2) \leftarrow \text{Proof}_2^{(I)(f.\text{E}_{\text{sk}})}(\text{vk}, \text{crs}, h, C, state_1), \\ tr^* \text{ denotes all oracle queries made so far,} \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr), \\ \beta \leftarrow_{\S} \{0, 1\}^m \\ \hat{\gamma} \leftarrow \text{Proof}_3^{(I)(f.\text{E}_{\text{sk}})}(C, \hat{x}, \hat{\alpha}, \beta, state_2). \end{array} \right] + \text{negl}(\lambda).
\end{aligned}$$

II. Prove that

$$\begin{aligned}
& \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.\text{E}_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{(f.\text{E}_{\text{sk}})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{f.\text{E}_{\text{sk}}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{(f.\text{E}_{\text{sk}})}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, td) \end{array} \right] \\
&\leq \text{negl}(\lambda).
\end{aligned}$$

Combining these will give us the desired result.

Part I. Here we will lower bound the probability of the event

$$\Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(I)(f.\text{Esk})}(\psi, \hat{x}, \hat{\alpha}, \beta, \hat{\gamma}) = 1 \\ \wedge C(x, w) = 0 \\ \wedge \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, \text{state}_1) \leftarrow \text{Proof}_1^{(I)(f.\text{Esk})}(\text{crs}, h), \\ \psi \leftarrow \text{Pre}^{(I)(f.\text{Esk})}(h, C), \\ (\hat{x}, \hat{\alpha}, \text{state}_2) \leftarrow \text{Proof}_2^{(I)(f.\text{Esk})}(\text{vk}, \text{crs}, h, C, \text{state}_1), \\ \text{tr}^* \text{ denotes all oracle queries made so far,} \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{state}_2, \text{tr}), \\ \beta \leftarrow_{\$} \{0, 1\}^m \\ \hat{\gamma} \leftarrow \text{Proof}_3^{(I)(f.\text{Esk})}(C, \hat{x}, \hat{\alpha}, \beta, \text{state}_2). \end{array} \right] \\ = \Pr[\mathbf{Hybrid 1}^{\mathcal{B}} = 1],$$

where **Hybrid 1** is defined as below.

Hybrid 1:

- $\text{crs} \leftarrow f.\text{Setup}(1^\lambda).$
- $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs}).$
- $h \leftarrow \text{Hash.Gen}(1^\lambda).$
- $\text{Proof}^{(I)(f.\text{Esk})}(\text{crs}, \text{vk}, h) :$
- $(\tilde{\text{crs}}, \tilde{\text{td}}) \leftarrow \text{SimSetup}(1^\lambda).$
- $(\tilde{\text{vk}}, \tilde{\text{sk}}) \leftarrow \text{SimGen}(1^\lambda, \tilde{\text{td}}, \tilde{\text{crs}}).$

Define the following oracle as E' that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$
- For i^{th} query q_i made by $\mathcal{B}$ to $f.\text{Esk}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $\tilde{\beta}_i \leftarrow_{\$} \{0, 1\}^m.$
 2. $(\tilde{\text{st}}_i, \tilde{\pi}_i) \leftarrow \text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \tilde{\beta}_i, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \tilde{\text{crs}}).$
 3. Store $(\psi_i, x_i, \alpha_i, \tilde{\text{st}}_i, \tilde{\pi}_i, \tilde{\beta}_i)$ in tr .
 4. Return $(\tilde{\text{st}}_i, \tilde{\pi}_i, \tilde{\beta}_i).$
 - * If $i = i^*$,
 1. Send $(\psi_{i^*}, x_{i^*}, \alpha_{i^*})$ as the first round message for the interactive proof.
 2. Let β_{i^*} be the challenge received.

3. $(\tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}) \leftarrow \text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \beta^*, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \tilde{\text{crs}})$.
4. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}, \beta_{i^*})$ to tr .
5. Let tr^* denotes all oracle queries made till the i^* th query,
6. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
7. Return $(\tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}, \beta_{i^*})$.

- $\text{Proof}_1^{(I)(f.\text{Esk})}(\text{crs}, \text{vk}, h)$:

1. Choose random $i^* \in [q]$.
2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{\text{E}'}(\tilde{\text{crs}}, \tilde{\text{vk}}, h)$.
3. Return C to the interactive verifier.
4. $state_1 := (\text{st}_1, tr)$

- $\psi \leftarrow \text{Pre}^{(I)(f.\text{Esk})}(h, C)$.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I)(f.\text{Esk})}(\text{crs}, \text{vk}, C, state_1, tr)$:

- $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_2) \leftarrow \mathcal{B}_2^{\text{E}'}(1^\lambda, \tilde{\text{crs}}, \tilde{\text{vk}}, C, \text{st}_1)$.
- If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \tilde{\text{st}}_{i^*})$, then abort.
- Output $\hat{\gamma}$ as the final message.

- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.
- Let $b \leftarrow \text{Verify}^{f.\text{Esk}, (I)}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$
- if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Let $(f.\text{SimSetup}, f.\text{SimGen}, f.\text{SimProve})$ be the stateful and simulatable VRF's simulator for $f := (f.\text{Setup}, f.\text{Gen}, f.\text{E})$. For every such simulator, we define a wrapper function $f.\text{SimEval}(\text{sk}, \text{td}, x, \text{st}, \text{crs})$ that does the following:

$f.\text{SimEval}(\text{sk}, \text{td}, x, \text{st}, \text{crs})$:

1. $y \leftarrow_{\$} \{0, 1\}^m$.
2. $(\pi, \text{st}') \leftarrow \text{SimProve}(\text{sk}, \text{td}, y, x, \text{st}, \text{crs})$.
3. Output (st', y, π) .

With this, we now define our second intermediate hybrid.

Hybrid 2: Same as **Hybrid 1**, except we replace $(f.\text{Setup}, f.\text{Gen}, f.\text{E})$ with the stateful and simulatable VRF's simulator for f , namely, $(f.\text{SimSetup}, f.\text{SimGen}, f.\text{SimEval})$, where $f.\text{SimEval}$ uses $f.\text{SimProve}$.

- $(\text{crs}, \text{td}) \leftarrow f.\text{SimSetup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{SimGen}(1^\lambda, \text{td}, \text{crs})$.
- $h \leftarrow \text{Hash.Gen}(1^\lambda)$.

$\text{Proof}^{(I),f.\text{SimEval}}(\text{crs}, \text{vk}, h) :$

- $(\tilde{\text{crs}}, \tilde{\text{td}}) \leftarrow f.\text{SimSetup}(1^\lambda).$
- $(\tilde{\text{vk}}, \tilde{\text{sk}}) \leftarrow f.\text{SimGen}(1^\lambda, \tilde{\text{td}}, \tilde{\text{crs}}).$

Define the following oracle as E' that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\tilde{\text{st}}_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to E' :
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $\tilde{\beta}_i \leftarrow_{\$} \{0, 1\}^m$.
 2. $(\tilde{\text{st}}_i, \tilde{\pi}_i) \leftarrow f.\text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \tilde{\beta}_i, (0^v, \psi_i, x_i, \alpha_i), \tilde{\text{st}}_{i-1}, \tilde{\text{crs}}).$
 3. Store $(\psi_i, x_i, \alpha_i, \tilde{\text{st}}_i, \tilde{\pi}_i, \tilde{\beta}_i)$ in tr .
 4. Return $(\tilde{\text{st}}_i, \tilde{\pi}_i, \tilde{\beta}_i)$.
 - * If $i = i^*$,
 1. Send $(\psi_{i^*}, x_{i^*}, \alpha_{i^*})$ as the first round message for the interactive proof.
 2. Let β_{i^*} be the challenge received.
 3. $(\tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}) \leftarrow f.\text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \beta_{i^*}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \tilde{\text{st}}_{i^*-1}, \tilde{\text{crs}}).$
 4. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}, \beta_{i^*})$ to tr .
 5. Let tr^* denote all oracle queries made till the i^* th query.
 6. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 7. Return $(\tilde{\text{st}}_{i^*}, \tilde{\pi}_{i^*}, \beta_{i^*})$.

- $\text{Proof}_1^{(I),f.\text{SimEval}}(\text{crs}, \text{vk}, h) :$

1. Choose random $i^* \in [q]$.
2. $(C, st_1) \leftarrow \mathcal{B}_1^{E'}(\tilde{\text{crs}}, \tilde{\text{vk}}, h).$
3. Return C to the interactive verifier.
4. $state_1 := (st_1, tr).$

- $\psi \leftarrow \text{Pre}^{(I),f.\text{SimEval}}(h, C).$

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I),f.\text{SimEval}}(\text{crs}, \text{vk}, C, state_1, tr) :$
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_{\text{out}}) \leftarrow \mathcal{B}_2^{E'}(1^\lambda, \tilde{\text{crs}}, \tilde{\text{vk}}, C, st_1).$
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \tilde{\text{st}}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.

- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{state}_2, \text{tr}^*)$.
- Let $b \leftarrow \text{Verify}^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
- if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Hybrid 1 and **Hybrid 2** are computationally indistinguishable assuming simulatability of f . In more detail, if there is an adversary that can distinguish between these two hybrids, then this adversary can be used to break the simulatability of f via a hybrid argument.

Lemma 5.5. *For all $\lambda \in \mathbb{N}$, assuming simulatability of f , there exists a negligible function $\text{negl}(\cdot)$ such that*

$$\left| \Pr[\text{Hybrid 1}^{\mathcal{B}} = 1] - \Pr[\text{Hybrid 2}^{\mathcal{B}} = 1] \right| \leq \text{negl}(\lambda).$$

Proof. For the sake of contradiction, if this is not true, then we can construct an adversary $\mathcal{D}$ that breaks the simulatability of f . Upon input crs , $\mathcal{D}$ makes a Gen query and receives vk . Note that $\mathcal{D}$ either gets crs and vk from the actual $f.\text{Setup}$ and $f.\text{Gen}$, and is given oracle access to $f.E$, or it gets crs and vk from the actual SimSetup and SimGen , and is given oracle access to SimEval .

$\mathcal{D}$ then runs all the steps of **Hybrid 1** (or interchangeably **Hybrid 2**) sequentially. Except it forwards all oracle queries to the evaluation function of f made during the execution of the hybrid by C , $\text{Verify}^{(I)}$, $\text{Proof}^{(I)}$ to its evaluation oracle. Observe that at this stage, $\text{Proof}^{(I)}$ is not making any oracle queries, rather it is simulating everything locally. It then gives the output of the hybrid evaluation to $\mathcal{D}$. If $\mathcal{B}$ has access to $f.E$, then it is running **Hybrid 1**, else it is running **Hybrid 2**. Therefore, the distinguishing advantage of $\mathcal{B}$ is identical to difference in the probability of the two hybrids outputting 1.

Hybrid 3: Identical to **Hybrid 2** except, $\text{Proof}^{(I)}$ uses the simulator for f directly to respond to oracle queries by $\mathcal{B}$.

- $(\text{crs}, \text{td}) \leftarrow f.\text{SimSetup}(1^\lambda)$.
 - $(\text{vk}, \text{sk}) \leftarrow f.\text{SimGen}(1^\lambda, \text{td}, \text{crs})$.
 - $h \leftarrow \text{Hash.Gen}(1^\lambda)$.
- $\text{Proof}^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, h) :$
- ~~$(\tilde{\text{crs}}, \tilde{\text{td}}) \leftarrow \text{SimSetup}(1^\lambda)$.~~
 - ~~$(\tilde{\text{vk}}, \tilde{\text{sk}}) \leftarrow \text{SimGen}(1^\lambda, \tilde{\text{td}}, \tilde{\text{crs}})$.~~

Define the following oracle as E' that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}$, $\text{Proof}_2^{(I)}$, $\text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to E' :
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,

1. $(\text{st}_i, \beta_i, \pi_i) \leftarrow f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. $\tilde{\beta}_i \leftarrow_{\$} \{0, 1\}^m$.
 3. $(\tilde{\text{st}}_i, \tilde{\pi}_i) \leftarrow \text{SimProve}(\tilde{\text{sk}}, \tilde{\text{td}}, \tilde{\beta}_i, (\psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 4. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 5. Return $(\text{st}_i, \pi_i, \beta_i)$.
- * If $i = i^*$,
1. Send $(\psi_{i^*}, x_{i^*}, \alpha_{i^*})$ as the first round message for the interactive proof.
 2. Let β_{i^*} be the challenge received.
 3. $(\text{st}_{i^*}, \pi_{i^*}) \leftarrow f.\text{SimProve}(\text{sk}, \text{td}, \beta_{i^*}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 4. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 5. Let tr^* denote all oracle queries made till the i^* th query.
 6. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 7. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{E'}(\text{crs}, \text{vk}, h)$.
 3. Return C to the interactive verifier.
 4. $state_1 := (\text{st}_1, tr)$.
 - $\psi \leftarrow \text{Pre}^{(I), f.\text{SimEval}}(h, C)$.
- We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.
- $\text{Proof}'^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, C, state_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_{\text{out}}) \leftarrow \mathcal{B}_2^{E'}(1^\lambda, \text{crs}, \text{vk}, C, \text{st}_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \text{st}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.
 - $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.
 - Let $b \leftarrow \text{Verify}^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
 - if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.6. Assuming state uniqueness of f and assuming that the padding parameter v is larger than the total size of $\text{Verify}^{(I)}$,

$$\left| \Pr[\text{Hybrid } 2^{\mathcal{B}} = 1] - \Pr[\text{Hybrid } 3^{\mathcal{B}} = 1] \right| \leq \text{negl}(\lambda).$$

Proof. The only difference between **Hybrid 2** and **Hybrid 3** is the way the oracle E' used to answer $\mathcal{B}$'s ssVRF queries is implemented.

First, the marginal view of $\mathcal{B}$ is identical in the two hybrids. Indeed, $(\tilde{\text{crs}}, \tilde{\text{vk}})$ and (crs, vk) are identically distributed. For every query $q_i = (0^v, \psi_i, x_i, \alpha_i)$ with $i \neq i^*$, the reply in both hybrids is generated by sampling a fresh uniform challenge and producing a simulated ssVRF proof for that challenge. For the guessed query $i = i^*$, the challenge is the interactive verifier's challenge β_{i^*} , which is also uniform over $\{0, 1\}^m$,

and the corresponding ssVRF proof is simulated. Hence the joint distribution of the public parameters given to $\mathcal{B}$ and all answers returned by E' is identical in the two hybrids.

It remains to argue that replacing the independently simulated instance used for E' by the outer simulated instance cannot be detected by the rest of the experiment. The circuit C has no oracle access in our model, and therefore cannot test whether the ssVRF outputs received by $\mathcal{B}$ are consistent with the outer oracle. The preprocessing algorithm is honestly executed on C , so its output is unchanged.

The only remaining possible distinguisher is $\text{Verify}^{(I)}$. Although the final message produced by $\mathcal{B}$ may contain strings that can be parsed as ssVRF states and proofs, every ssVRF query made by $\mathcal{B}$ is in the Fiat–Shamir domain and therefore has the form $q_i = (0^v, \psi_i, x_i, \alpha_i)$. Since v is chosen to be larger than the total size of $\text{Verify}^{(I)}$, the verifier cannot explicitly form any such query string. Consequently, $\text{Verify}^{(I)}$ cannot query $f.\text{SimEval}$ on any of the points queried by $\mathcal{B}$, nor can it locally run $f.\text{Verify}$ on any tuple whose VRF input is one of these padded points. This remains true even if the prover’s final message contains the corresponding state and proof, because invoking $f.\text{Verify}$ requires explicitly supplying the full padded input.

For any oracle query made by $\text{Verify}^{(I)}$ outside the Fiat–Shamir padded domain, the answer is generated by the same outer oracle in both hybrids. Thus such queries have identical distributions in the two hybrids. Moreover, by state uniqueness, even if an admissible verifier query collided with a point previously queried through E' , it would not reproduce the same state used in $\mathcal{B}$ ’s reply except with negligible probability.

Therefore, conditioned on the negligible state-collision event not occurring, the entire view of $\text{Verify}^{(I)}$ and the output predicate of the hybrid experiment are the same in **Hybrid 2** and **Hybrid 3**. This finishes the proof.

Hybrid 4: We replace oracle access to $f.\text{SimEval}$ for $\text{Verify}^{(I)}$ and $\text{Pre}^{(I)}$ with oracle access to E' .

- $(\text{crs}, \text{td}) \leftarrow f.\text{SimSetup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{SimGen}(1^\lambda, \text{td}, \text{crs})$.
- $h \leftarrow \text{Hash.Gen}(1^\lambda)$.

$\text{Proof}^{(I), E'}(\text{crs}, \text{vk}, h) :$

Define the following oracle as E' that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle’s input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}$, $\text{Proof}_2^{(I)}$, $\text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to E' :
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $(\text{st}_i, \beta_i, \pi_i) \leftarrow f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 3. Return $(\text{st}_i, \pi_i, \beta_i)$.
 - * If $i = i^*$,
 1. Send $(\psi_{i^*}, x_{i^*}, \alpha_{i^*})$ as the first round message for the interactive proof.
 2. Let β_{i^*} be the challenge received.

3. $(\text{st}_{i^*}, \pi_{i^*}) \leftarrow f.\text{SimProve}(\text{sk}, \text{td}, \beta_{i^*}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
4. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
5. Let tr^* denote all oracle queries made till the i^* th query.
6. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
7. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.

• $\text{Proof}_1^{(I), E'}(\text{crs}, \text{vk}, h)$:

1. Choose random $i^* \in [q]$.
2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{E'}(\text{crs}, \text{vk}, h)$.
3. Return C to the interactive verifier.
4. $state_1 := (\text{st}_1, tr)$.

• $\psi \leftarrow \text{Pre}^{(I), E'}(h, C)$.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I), E'}(\text{crs}, \text{vk}, C, state_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_{\text{out}}) \leftarrow \mathcal{B}_2^{E'}(1^\lambda, \text{crs}, \text{vk}, C, \text{st}_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \text{st}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.
- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.
- Let $b \leftarrow \text{Verify}^{(I), E'}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
- if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.7. Assuming state uniqueness and proof unforgeability of $(f.\text{SimSetup}, f.\text{SimGen}, f.\text{SimEval})$,

$$\left| \Pr[\text{Hybrid } 3^B = 1] - \Pr[\text{Hybrid } 4^B = 1] \right| \leq \text{negl}(\lambda).$$

Proof. The only difference between these hybrids is whether β_{i^*} sent during the interactive proof is programmed as an output to $(0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*})$ with state st_{i^*} or not. Observe that the verifier can again not run a verification query on the output of $\mathcal{B}$ because of the padding. The distinguishing advantage of these two hybrids is bounded by the distinguishing probability between two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, given access to $\text{crs}, \text{vk}, \beta_{i^*}$, where $\mathcal{D}_1$ is the distribution of the tuple

$$(0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}, E'(0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*})),$$

and $\mathcal{D}_2$ is the distribution of the tuple

$$(0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}, f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})).$$

Observe that $f.\text{SimEval}$ and E' are identical except at the programmed tuple $(0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$. Therefore, the only way one can distinguish between these two distributions is to either

1. Query the evaluation oracle $f.\text{SimEval}$ at $(0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*})$, and the simulator picks st_{i^*} as its state, or

2. The above event does not happen but one can find (st_{i^*}, π_{i^*}) and check if

$$f.\text{Verify}(\text{vk}, ((0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), st_{i^*}), \beta_{i^*}, \pi_{i^*}, \text{crs}) = 1.$$

Let us call these events E_1 and E_2 respectively. Now,

$$\left| \Pr[\mathbf{Hybrid\ 3}^{\mathcal{B}} = 1] - \Pr[\mathbf{Hybrid\ 4}^{\mathcal{B}} = 1] \right| \leq \Pr[E_1] + \Pr[E_2 \mid \neg E_1].$$

Here, the first probability is bounded above by a negligible function because of the state uniqueness property of f in the simulation world. For every PPT adversary, the second event is also bounded above by a negligible function due to proof unforgeability of f in the simulated world. Therefore,

$$\left| \Pr[\mathbf{Hybrid\ 3}^{\mathcal{B}} = 1] - \Pr[\mathbf{Hybrid\ 4}^{\mathcal{B}} = 1] \right| \leq \text{negl}(\lambda).$$

Hybrid 5: We change the way β_{i^*} is sampled.

- $(\text{crs}, \text{td}) \leftarrow f.\text{SimSetup}(1^\lambda).$
- $(\text{vk}, \text{sk}) \leftarrow f.\text{SimGen}(1^\lambda, \text{td}, \text{crs}).$
- $h \leftarrow \text{Hash.Gen}(1^\lambda).$

$\text{Proof}^{(I), E'}(\text{crs}, \text{vk}, h) :$

Define the following oracle as E' that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}$, $\text{Proof}_2^{(I)}$, $\text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $st_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to E' :
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $(st_i, \beta_i, \pi_i) \leftarrow f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_i, x_i, \alpha_i), st_{i-1}, \text{crs}).$
 2. Store $(\psi_i, x_i, \alpha_i, st_i, \pi_i, \beta_i)$ in tr .
 3. Return (st_i, π_i, β_i) .
 - * If $i = i^*$,
 1. ~~Send $(\psi_{i^*}, x_{i^*}, \alpha_{i^*})$ as the first round message for the interactive proof.~~
 2. ~~Let β_{i^*} be the challenge received.~~
 3. $(st_{i^*}, \beta_{i^*}, \pi_{i^*}) \leftarrow f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), st_{i^*-1}, \text{crs}).$
 4. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, st_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 5. Let tr^* denote all oracle queries made till the i^* th query.
 6. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 7. Return $(st_{i^*}, \pi_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I), E'}(\text{crs}, \text{vk}, h) :$

1. Choose random $i^* \in [q]$.
 2. $(C, st_1) \leftarrow \mathcal{B}_1^{E'}(\text{crs}, \text{vk}, h)$.
 3. Return C to the interactive verifier.
 4. $state_1 := (st_1, tr)$.
- $\psi \leftarrow \text{Pre}^{(I), E'}(h, C)$.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I), E'}(\text{crs}, \text{vk}, C, state_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{st}, \hat{\pi}, st_{\text{out}}) \leftarrow \mathcal{B}_2^{E'}(1^\lambda, \text{crs}, \text{vk}, C, st_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{st}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, st_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.
- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.
- Let $b \leftarrow \text{Verify}^{(I), E'}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
- if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.8. Assuming state uniqueness and proof unforgeability of $(f.\text{SimSetup}, f.\text{SimGen}, f.\text{SimEval})$,

$$\left| \Pr[\text{Hybrid } 4^B = 1] - \Pr[\text{Hybrid } 5^B = 1] \right| = 0.$$

Proof. In both the hybrids, the distribution of $(\beta_{i^*}, st_{i^*}, \pi_{i^*})$ remain identical, and therefore the view of $\mathcal{B}$ remains unchanged. Thus, the two hybrids are identical if $\text{Verify}^{(I)}$ does not make oracle query at the programmed point. This follows from arguments identical to the proof of the previous lemma.

Hybrid 6: We rename the oracle E' to $f.\text{SimEval}$ as they are now identical.

- $(\text{crs}, \text{td}) \leftarrow f.\text{SimSetup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{SimGen}(1^\lambda, \text{td}, \text{crs})$.
- $h \leftarrow \text{Hash.Gen}(1^\lambda)$.

$\text{Proof}^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, h) :$

Define the following oracle as $f.\text{SimEval}$ that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $st_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to $f.\text{SimEval}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else

- * If $i \neq i^*$,
 1. $(\text{st}_i, \beta_i, \pi_i) \leftarrow f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 3. Return $(\text{st}_i, \pi_i, \beta_i)$.
 - * If $i = i^*$,
 1. $(\text{st}_{i^*}, \beta_{i^*}, \pi_{i^*}) \leftarrow f.\text{SimEval}(\text{sk}, \text{td}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 2. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 3. Let tr^* denote all oracle queries made till the i^* th query.
 4. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 5. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.
 - $\text{Proof}_1^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{f.\text{SimEval}}(\text{crs}, \text{vk}, h)$.
 3. Return C to the interactive verifier.
 4. $state_1 := (\text{st}_1, tr)$.
 - $\psi \leftarrow \text{Pre}^{(I), f.\text{SimEval}}(h, C)$.
- We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.
- $\text{Proof}'^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, C, state_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_{\text{out}}) \leftarrow \mathcal{B}_2^{f.\text{SimEval}}(1^\lambda, \text{crs}, \text{vk}, C, \text{st}_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \text{st}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.
 - $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.
 - Let $b \leftarrow \text{Verify}^{(I), f.\text{SimEval}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
 - if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.9.

$$\left| \Pr[\text{Hybrid } 5^B = 1] - \Pr[\text{Hybrid } 6^B = 1] \right| = 0.$$

This is simply a renaming of the oracles. The hybrids are indeed identical.

Hybrid 7: We replace $f.\text{SimEval}$ with the real world ssVRF f .

- $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$.
- $h \leftarrow \text{Hash.Gen}(1^\lambda)$.

$\text{Proof}^{(I), f.E_{\text{sk}}}(\text{crs}, \text{vk}, h) :$

Define the following oracle as $f.E_{\text{sk}}$ that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}$, $\text{Proof}_2^{(I)}$, $\text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to $f.\text{E}_{\text{sk}}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $(\beta_i, \pi_i, \text{st}_i) \leftarrow f.\text{E}(\text{sk}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 3. Return $(\text{st}_i, \pi_i, \beta_i)$.
 - * If $i = i^*$,
 1. $(\beta_{i^*}, \pi_{i^*}, \text{st}_{i^*}) \leftarrow f.\text{E}(\text{sk}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 2. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 3. Let tr^* denote all oracle queries made till the i^* th query.
 4. Let state_2 be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 5. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I), f.\text{E}_{\text{sk}}}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{f.\text{E}_{\text{sk}}}(\text{crs}, \text{vk}, h)$.
 3. Return C to the interactive verifier.
 4. $\text{state}_1 := (\text{st}_1, tr)$.
- $\psi \leftarrow \text{Pre}^{(I), f.\text{E}_{\text{sk}}}(h, C)$.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I), f.\text{E}_{\text{sk}}}(\text{crs}, \text{vk}, C, \text{state}_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_{\text{out}}) \leftarrow \mathcal{B}_2^{f.\text{E}_{\text{sk}}}(1^\lambda, \text{crs}, \text{vk}, C, \text{st}_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \text{st}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.
- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{state}_2, tr^*)$.
- Let $b \leftarrow \text{Verify}^{(I), f.\text{E}_{\text{sk}}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
- if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.10. For all $\lambda \in \mathbb{N}$, assuming simulatability of f , there exists a negligible function $\text{negl}(\cdot)$ such that

$$\left| \Pr[\text{Hybrid } 6^{\mathcal{B}} = 1] - \Pr[\text{Hybrid } 7^{\mathcal{B}} = 1] \right| \leq \text{negl}(\lambda).$$

The proof is identical to that of the indistinguishability of **Hybrid 1** and **Hybrid 2**.

Hybrid 8: We add a new abort statement using the ssVRF verification on the challenge output.

- $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$.
- $h \leftarrow \text{Hash}.\text{Gen}(1^\lambda)$.

$\text{Proof}^{(I), f, \text{E}_{\text{sk}}}(\text{crs}, \text{vk}, h)$:

Define the following oracle as $f.\text{E}_{\text{sk}}$ that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}$, $\text{Proof}_2^{(I)}$, $\text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to $f.\text{E}_{\text{sk}}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $(\beta_i, \pi_i, \text{st}_i) \leftarrow f.\text{E}(\text{sk}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 3. Return $(\text{st}_i, \pi_i, \beta_i)$.
 - * If $i = i^*$,
 1. $(\beta_{i^*}, \pi_{i^*}, \text{st}_{i^*}) \leftarrow f.\text{E}(\text{sk}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 2. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 3. Let tr^* denote all oracle queries made till the i^* th query.
 4. Let state_2 be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 5. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I), f, \text{E}_{\text{sk}}}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{f, \text{E}_{\text{sk}}}(\text{crs}, \text{vk}, h)$.
 3. Return C to the interactive verifier.
 4. $\text{state}_1 := (\text{st}_1, tr)$.
- $\psi \leftarrow \text{Pre}^{(I), f, \text{E}_{\text{sk}}}(h, C)$.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I), f, \text{E}_{\text{sk}}}(\text{crs}, \text{vk}, C, \text{state}_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_{\text{out}}) \leftarrow \mathcal{B}_2^{f, \text{E}_{\text{sk}}}(1^\lambda, \text{crs}, \text{vk}, C, \text{st}_1)$.
 - If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, \text{st}_{i^*})$, then abort.
 - Output $\hat{\gamma}$ as the final message.

- If $f.\text{Verify}(\text{vk}, ((0^v, \psi, \hat{x}, \hat{\alpha}), \hat{\text{st}}), \hat{\beta}, \hat{\pi}, \text{crs}) \neq 1$, then abort.
- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{state}_2, \text{tr}^*)$.
- Let $b \leftarrow \text{Verify}^{(I), f.E_{\text{sk}}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
- if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.11.

$$\Pr[\text{Hybrid } 7^{\mathcal{B}} = 1] \geq \Pr[\text{Hybrid } 8^{\mathcal{B}} = 1]$$

Proof. Hybrid 8 is identical to Hybrid 7 with an additional abort statement. Therefore, its success probability can only decrease.

Hybrid 9: We remove the older abort statement.

- $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$.
- $h \leftarrow \text{Hash}.\text{Gen}(1^\lambda)$.

$\text{Proof}^{(I), f.E_{\text{sk}}}(\text{crs}, \text{vk}, h)$:

Define the following oracle as $f.E_{\text{sk}}$ that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}$, $\text{Proof}_2^{(I)}$, $\text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $\text{st}_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to $f.E_{\text{sk}}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,
 1. $(\beta_i, \pi_i, \text{st}_i) \leftarrow f.E(\text{sk}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 3. Return $(\text{st}_i, \pi_i, \beta_i)$.
 - * If $i = i^*$,
 1. $(\beta_{i^*}, \pi_{i^*}, \text{st}_{i^*}) \leftarrow f.E(\text{sk}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 2. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 3. Let tr^* denote all oracle queries made till the i^* th query.
 4. Let state_2 be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 5. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I), f.E_{\text{sk}}}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{f.E_{\text{sk}}}(\text{crs}, \text{vk}, h)$.

3. Return C to the interactive verifier.
 4. $state_1 := (st_1, tr)$.
- $\psi \leftarrow \text{Pre}^{(I),f.\text{Esk}}(h, C)$.
- We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.
- $\text{Proof}'^{(I),f.\text{Esk}}(\text{crs}, \text{vk}, C, state_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_{\text{out}}) \leftarrow \mathcal{B}_2^{f.\text{Esk}}(1^\lambda, \text{crs}, \text{vk}, C, st_1)$.
 - ~~If $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\text{st}}) \neq (x_{i^*}, \alpha_{i^*}, \beta_{i^*}, st_{i^*})$, then abort.~~
 - Output $\hat{\gamma}$ as the final message.
 - If $f.\text{Verify}(\text{vk}, ((0^v, \psi, \hat{x}, \hat{\alpha}), \hat{\beta}, \hat{\pi}, \text{crs}) \neq 1$, then abort.
 - $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.
 - Let $b \leftarrow \text{Verify}^{(I),f.\text{Esk}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
 - if $C \in \mathcal{C}$, $b == 1 \wedge C(\hat{x}, w) = 0$, then output 1.

Lemma 5.12.

$$\Pr[\text{Hybrid } 9^{\mathcal{B}} = 1] = q \Pr[\text{Hybrid } 8^{\mathcal{B}} = 1].$$

Proof. Since, we are assuming that Good is true, this implies that $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$ is a part of the input-output string for some oracle query made by $\mathcal{B}$. Since i^* is picked uniformly at random, the probability that the abort does not take place is exactly $\frac{1}{q}$.

Hybrid 10: The extractor now uses the entire transcript to generate the witness rather than the partial transcript.

- $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$.
- $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$.
- $h \leftarrow \text{Hash}.\text{Gen}(1^\lambda)$.

$\text{Proof}^{(I),f.\text{Esk}}(\text{crs}, \text{vk}, h)$:

Define the following oracle as $f.\text{Esk}$ that is used by $\text{Proof}^{(I)}$ to respond to oracle queries made by $\mathcal{B}$:

Remark: This oracle's input-output can be stored in the honestly maintained state tr during the execution. Using information in tr , all the three prover algorithms ($\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)}$) can respond to queries made to this oracle in a consistent manner.

- Initialize $st_0 = \perp$.
- For the i^{th} query q_i made by $\mathcal{B}$ to $f.\text{Esk}$:
 - If q_i cannot be parsed as $(0^v, \psi_i, x_i, \alpha_i)$, then abort.
 - Else
 - * If $i \neq i^*$,

1. $(\beta_i, \pi_i, \text{st}_i) \leftarrow f.\text{E}(\text{sk}, (0^v, \psi_i, x_i, \alpha_i), \text{st}_{i-1}, \text{crs})$.
 2. Store $(\psi_i, x_i, \alpha_i, \text{st}_i, \pi_i, \beta_i)$ in tr .
 3. Return $(\text{st}_i, \pi_i, \beta_i)$.
- * If $i = i^*$,
1. $(\beta_{i^*}, \pi_{i^*}, \text{st}_{i^*}) \leftarrow f.\text{E}(\text{sk}, (0^v, \psi_{i^*}, x_{i^*}, \alpha_{i^*}), \text{st}_{i^*-1}, \text{crs})$.
 2. Add $(\psi_{i^*}, x_{i^*}, \alpha_{i^*}, \text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$ to tr .
 3. Let tr_{i^*} denote all oracle queries made till the i^* th query.
 4. Let $state_2$ be the auxiliary information maintained by $\text{Proof}'^{(I)}$ until this point.
 5. Return $(\text{st}_{i^*}, \pi_{i^*}, \beta_{i^*})$.
- $\text{Proof}_1^{(I), f.\text{Esk}}(\text{crs}, \text{vk}, h)$:
 1. Choose random $i^* \in [q]$.
 2. $(C, \text{st}_1) \leftarrow \mathcal{B}_1^{f.\text{Esk}}(\text{crs}, \text{vk}, h)$.
 3. Return C to the interactive verifier.
 4. $state_1 := (\text{st}_1, tr)$.
 - $\psi \leftarrow \text{Pre}^{(I), f.\text{Esk}}(h, C)$.

We use Proof' to denote the combination of $\text{Proof}_2^{(I)}$ and $\text{Proof}_3^{(I)}$.

- $\text{Proof}'^{(I), f.\text{Esk}}(\text{crs}, \text{vk}, C, state_1, tr)$:
 - $(\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_{\text{out}}) \leftarrow \mathcal{B}_2^{f.\text{Esk}}(1^\lambda, \text{crs}, \text{vk}, C, \text{st}_1)$.
 - Output $\hat{\gamma}$ as the final message.
- If $f.\text{Verify}(\text{vk}, ((0^v, \psi, \hat{x}, \hat{\alpha}), \hat{\text{st}}), \hat{\beta}, \hat{\pi}, \text{crs}) \neq 1$, then abort.
- $w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{st}_2, tr)$.
- Let $b \leftarrow \text{Verify}^{(I), f.\text{Esk}}(\text{crs}, \text{vk}, \psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$.
- If $C \in \mathcal{C}$, $b == 1$, and $C(\hat{x}, w) = 0$, then output 1.

Lemma 5.13. *Assuming Good takes place,*

$$\Pr[\text{Hybrid } 10^B = 1] = \Pr[\text{Hybrid } 9^B = 1].$$

Proof. If Good occurs, then we know that $E^{(I)}(C, \hat{x}, \hat{\alpha}, \text{st}_2, tr) = E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$, and the lemma follows.

Finally observe that by construction,

$$\begin{aligned} & \Pr[\text{Hybrid } 10^B = 1] \\ &= \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.\text{Esk})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \text{Good} \end{array} \right] \left[\begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, \text{st}_1) \leftarrow \mathcal{B}_1^{f.\text{Esk}}(\text{crs}, \text{vk}, h), \\ \psi \leftarrow \text{Pre}^{(f.\text{Esk})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_2) \leftarrow \mathcal{B}_2^{(f.\text{Esk})}(\text{vk}, \text{crs}, h, C, \text{st}_1), \\ tr := \text{all queries made to the oracle}, \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, \text{st}_2, tr) \end{array} \right] \end{aligned}$$

Combining these together, we conclude the proof of Part I, i.e.,

$$\begin{aligned}
& \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(f.E_{\text{sk}})}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f.E_{\text{sk}}}(\text{crs}, \text{vk}, h), \\ \psi \leftarrow \text{Pre}^{(f.E_{\text{sk}})}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f.E_{\text{sk}}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
& \leq q \cdot \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(I)(f.E_{\text{sk}})}(\psi, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}) = 1 \\ \wedge C(\hat{x}, w) = 0 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, state_1) \leftarrow \text{Proof}_1^{(f.E_{\text{sk}})}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{(I), f.E_{\text{sk}}}(h, C), \\ (\hat{x}, \hat{\alpha}, state_2) \leftarrow \text{Proof}_2^{(I)(f.E_{\text{sk}})}(\text{vk}, \text{crs}, h, C, state_1), \\ tr \text{ denotes all oracle queries made so far,} \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr), \\ \hat{\beta} \leftarrow_{\S} \{0, 1\}^m \\ \hat{\gamma} \leftarrow \text{Proof}_3^{(I)(f.E_{\text{sk}})}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, state_2). \end{array} \right] + \text{negl}(\lambda).
\end{aligned}$$

Part II. To begin with, let us split the event Good into two distinct events:

1. G_1 : Let $\text{Pre}^{(f.E_{\text{sk}})}(C) = \psi \wedge (\psi, \hat{x}, \hat{\alpha})$ was used by $\mathcal{B}$ as an oracle query to $f.E$ and the response was $(\hat{\text{st}}, \hat{\beta}, \hat{\pi})$ and
2. G_2 : $E^{(I)}(C, \hat{x}, \hat{\alpha}, st_2, tr) = E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*)$.

Thus $\text{Good} = G_1 \wedge G_2$.

Now,

$$\begin{aligned}
& \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg \text{Good} \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&= \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge (\neg G_1 \vee \neg G_2) \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&\leq \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&+ \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg G_2 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&\leq \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&\quad + \Pr \left[\begin{array}{l} \neg G_2 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right]
\end{aligned}$$

Lemma 5.14. Assuming knowledge soundness of the relativized interactive proof system $(\text{Setup}, \text{Pre}^{(I)}, \text{Proof}^{(I)}, \text{Verify}^{(I)})$,

$$\Pr \left[\begin{array}{l} \neg G_2 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \leq \text{negl}(\lambda).$$

Proof. Observe that by construction,

$$\begin{aligned}
& \Pr \left[\neg G_2 \mid \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\
&= \Pr \left[\neg G_2 \mid \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, st_2, tr) \end{array} \right] \\
&= \Pr \left[\neg G_2 \mid \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \text{Proof}_1^{(I), f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, state_2) \leftarrow \text{Proof}_2^{(I), f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ tr^* := \text{all queries made to the oracle so far}, \\ w' \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, st_2, tr) \end{array} \right] \\
&= \Pr \left[w \neq w' \mid \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \text{Proof}_1^{(I), f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, state_2) \leftarrow \text{Proof}_2^{(I), f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ tr^* := \text{all queries made to the oracle so far}, \\ w' \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, st_2, tr) \end{array} \right]
\end{aligned}$$

Now recall the definition of round by round knowledge soundness of the interactive proof. There exists an extractor $E^{(I)}$ such that for every malicious prover Proof and PPT adversary $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge w \neq w' \end{array} \mid \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \text{Proof}_1^{(I), f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, state_2) \leftarrow \text{Proof}_2^{(I), f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ tr^* := \text{all queries made to the oracle so far}, \\ w' \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, state_2, tr^*), \\ (\perp, st_2) \leftarrow \mathcal{A}^{f, \text{Esk}}(state_2) \\ tr := \text{all queries made to the oracle by } \mathcal{A}, \\ w \leftarrow E^{(I)}(C, \hat{x}, \hat{\alpha}, st_2, tr) \end{array} \right] \leq \text{negl}(\lambda).$$

This directly implies the lemma.

Lemma 5.15. Assuming unforgeability of ssVRF,

$$\Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \leq \text{negl}(\lambda).$$

Proof. Note that

$$\begin{aligned} & \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge C(\hat{x}, w) = 0 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} \text{Verify}^{f, \text{Esk}}(\psi, \hat{x}, (\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi})) = 1 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} f.V(\text{vk}, ((\psi, \hat{x}, \hat{\alpha}), \hat{\text{st}}), \hat{\beta}, \hat{\pi}; \text{crs}) = 1 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \end{aligned}$$

where the last inequality is due to the construction of Verify. The event G_1 does not occur if

1. $(\psi, \hat{x}, \hat{\alpha}, \hat{\text{st}}, \hat{\beta}, \hat{\pi})$ was not in tr .

Thus,

$$\begin{aligned} & \Pr \left[\begin{array}{l} f.V(\text{vk}, ((\psi, \hat{x}, \hat{\alpha}), \hat{\text{st}}), \hat{\beta}, \hat{\pi}; \text{crs}) = 1 \\ \wedge \neg G_1 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} f.V(\text{vk}, ((\psi, \hat{x}, \hat{\alpha}), \hat{\text{st}}), \hat{\beta}, \hat{\pi}; \text{crs}) = 1 \\ \wedge (\psi, \hat{x}, \hat{\alpha}, \hat{\text{st}}, \hat{\beta}, \hat{\pi}) \notin tr \end{array} \middle| \begin{array}{l} (\text{crs}, \text{vk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, st_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{f, \text{Esk}}(h, C), \\ (\hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, st_1), \\ w \leftarrow \mathcal{E}(C, \hat{x}, \hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\text{st}}, \hat{\pi}, st_2, tr) \end{array} \right] \end{aligned}$$

If the last probability is not negligible, then $\mathcal{B}$ would be a PPT adversary that can successfully break the unforgeability of f by using $(\psi, \hat{x}, \hat{\alpha}, \hat{\text{st}}, \hat{\beta}, \hat{\pi})$ as the challenge.

This completes the proof.

6 Constructions of Stateful and Simulatable VRF

We demonstrate a simple and generic method to convert any simulatable VRF to a stateful and simulatable one in a black-box manner. This gives us concrete instantiations of ssVRF via prior constructions of simulatable VRF.

6.1 Black-box construction of ssVRF from sVRF

Let $f.\text{Update}$ denote a deterministic algorithm that takes as input a state st , and outputs another state st' . We say that the state update mechanism is admissible if it satisfies the state uniqueness property in Definition 4.1, i.e., it outputs a unique state upto any polynomially many evaluations. Given a simulatable VRF f with algorithms $(f.\text{Setup}, f.\text{Gen}, f.\text{E}, f.\text{V})$, we show how to construct a stateful and simulatable VRF f' with algorithms $(f'.\text{Setup}, f'.\text{Gen}, f'.\text{E}, f'.\text{V})$ defined as follows with the internal state initialized as $\mathbf{0}$:

- $f'.\text{Setup}(1^\lambda)$:
 - $(\text{crs}, \text{td}) \leftarrow f.\text{Setup}(1^\lambda)$.
 - $\text{crs}' := \text{crs}, \text{td}' := \text{td}$.
- $f'.\text{Gen}(\text{crs}')$:
 - $(\text{sk}, \text{vk}) \leftarrow f.\text{Gen}(\text{crs})$.
 - $\text{vk}' := \text{vk}, \text{sk}' := \text{sk}$.
- $f'.\text{E}(\text{sk}', x; \text{st}, \text{crs}')$:
 - $\text{st}' \leftarrow f.\text{Update}(\text{st})$.
 - $(y, \pi) \leftarrow f.\text{E}(\text{sk}, (x, \text{st}'); \text{crs})$.
 - Update $\text{st} \leftarrow \text{st}'$.
 - Output (st', y, π) .
- $f'.\text{V}(\text{vk}', (x, \text{st}), (y, \pi); \text{crs}')$:
 - Return $f.\text{V}(\text{vk}, (x, \text{st}), (y, \pi); \text{crs})$.

Lemma 6.1. *Assuming f is an sVRF and $f.\text{Update}$ outputs unique values when evaluated for polynomially many times, the, f' described in construction above is an ssVRF as per Definition 4.1.*

Proof. The correctness and uniqueness of f' follows directly from the correctness of f . Uniqueness of $f.\text{Update}$ implies state uniqueness of f' .

For simulatability, the simulator for f' is identical to that of f , except that it additionally runs $f.\text{Update}$ locally while generating the simulated proofs.

Proof unforgeability of f' follows from a reduction from the unforgeability of f . Specifically, we can use an adversary $\mathcal{A}$ that can forge proofs for f' to another adversary $\mathcal{B}$ that can break unforgeability of f . First $\mathcal{B}$ passes crs, vk sent by the setup of f to $\mathcal{A}$ and sets an initial state st . Now, when $\mathcal{A}$ makes an evaluation query x , $\mathcal{B}$ first runs the state update algorithm $\text{st}' \leftarrow f.\text{Update}(\text{st})$ and forwards query (x, st') to the evaluation oracle $f.\text{E}$. It receives (y, π) as output and forwards (y, π, st') to $\mathcal{A}$. At the end of this interaction, $\mathcal{B}$ returns a tuple $(x^*, y^*, \text{st}^*, \pi^*)$ as a purported forgery. $\mathcal{A}$ parses this as $((x^*, \text{st}^*), y^*, \pi^*)$ and outputs this as its forgery. The probability of $\mathcal{B}$ winning the forgeability game of f is identical to that of $\mathcal{A}$ forgeability game of f' .

Instantiating $f.\text{Update}$. Since we only require uniqueness from the state update, this can be implemented in several ways:

- **Counter:** One can treat the state as a counter, i.e., $f.\text{Update}$ takes as input st and outputs $st + 1$.
- **PRF:** One can also use a PRF to update the state. Let PRF be a pseudorandom function with the following two algorithms: $(\text{PRF.Setup}, \text{PRF.E})$. Then, the PRF key generated by PRF.Setup can be a part of the ssVRF secret key and $f.\text{Update}$ takes as input the PRF key and current state st , and outputs new state as the evaluation of PRF on st . Pseudorandomness implies that with all but negligible probability, the states will remain unique for polynomial many evaluations.

Constructions of simulatable VRF. The construction of sVRF proposed by Chase and Lysyanskaya [CL07] (an extension of the efficient VRF construction by Dodis and Yampolskiy [DY05]) is an ideal candidate for our application. In particular, this construction *does not* rely on random oracles and can be based on standard group assumptions. Additionally there are several other instantiations of simulatable VRFs in deployed in real-world systems [Res23, Lab22, HMW18, GHK⁺25] that rely on random oracles. Using these constructions does mean that we have to use the random oracle heuristic to rely on security, however, they allow for more efficient implementations.

Instantiating stateful and simulatable VRF. There are several instantiations of simulatable VRF that already satisfy unique state property, and therefore can be directly used as a stateful and simulatable VRF. For instance, Supra’s dVRF [Res23] maintains a counter via smart-contract that can be adapted as the state for a ssVRF. Similarly, Aptos Roll [TX24] assigns a unique block number and transaction id in the block as the counter. In Chainlink’s VRF, every VRF request is indexed using a unique “RequestID” [cha25] that can be used as a state in our application of ssVRF.

7 Fiat-Shamir transformation in the VRF oracle model and application to Recursive Proofs

Now, we will show that the Fiat-Shamir transformation applied to non-relativized sigma protocols is also provably secure if instantiated with a simulatable VRF, i.e., replace the random oracle with the evaluation function of sVRF. Recall that in the standard Fiat-Shamir transformation, the sigma protocol is for a plain (non-relativized or oracle independent) language, and parties do not have access to any oracles. However, after Fiat-Shamir, the non-interactive proof system is in the VRF oracle model, i.e., all parties are allowed oracle access to the sVRF evaluation function.

Remark 7.1. We no longer need statefulness from the VRF for this construction as the circuit is no longer given access to the sVRF oracle.

This result can be thought of as (1) a provable and concrete instantiation the Fiat-Shamir transformation. (Recall that the Fiat-Shamir transformation with a concrete instantiation of the random oracle is known to be not sound in general), and (2) an alternate to the Correlation Intractable hash approach to yield concrete constructions of provably secure non-interactive proof systems.

Intuitively, if we used a stateful and simulatable VRF oracle here, this would have been a strictly weaker result than Theorem 5.2, and indeed would follow from that theorem in a straightforward manner. The proof involving the weaker notion of a simulatable VRF is also almost-identical. We formally write down the construction and give a high level overview of the proof idea for the sake of completeness. The application to recursive proofs is the interesting contribution here.

Theorem 7.2. *Assume*

- $(f.\text{Setup}, f.\text{Gen}, f.\text{E}, f.\text{V})$ is an sVRF satisfying all properties of Definition 3.4, and

- $(\text{Pre}^{(I)}, \text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Verify}^{(I)})$ is a sigma protocol that satisfies adaptive round-by-round knowledge soundness.

Then, for all $\lambda \in \mathbb{N}$, Construction 2 is a non-interactive argument system with adaptive knowledge soundness.

Construction 2 (Fiat-Shamir relative to sVRF).

- $\text{Setup}(1^\lambda)$:
 - $(\text{crs}, \text{td}) \leftarrow f.\text{Setup}(1^\lambda)$.
 - $(\text{vk}, \text{sk}) \leftarrow f.\text{Gen}(1^\lambda, \text{crs})$.
 - $h \leftarrow_{\$} \text{Hash.Gen}(1^\lambda)$.
 - Output $(\text{crs}, \text{vk}, h)$.
- $\text{Pre}(\text{crs}, \text{vk}, h, C)$:
 - $\psi \leftarrow \text{Pre}^{(I)}(h, C)$.
- $\text{Proof}^{f.E_{sk}}(\text{crs}, \text{vk}, h, C, \psi, x, w)$:
 - $\alpha \leftarrow \text{Proof}_1^{(I)}(h, C, \psi, x, w)$.
 - $(y, \text{st}', \pi) \leftarrow f.E_{sk}((\psi, x, \alpha), \text{crs}, \text{st})$.
 - $\beta := y$.
 - $\gamma \leftarrow \text{Proof}_2^{(I)}(C, \psi, x, w, \alpha, \beta)$.
 - $\Pi := (\alpha, \beta, \gamma, \text{st}', \pi)$.
- $\text{Verify}^{f.E_{sk}}(\text{crs}, \text{vk}, h, \psi, x, \Pi)$:
 - Parse Π as $(\alpha, \beta, \gamma, \text{st}, \pi)$.
 - If $f.V(\text{vk}, ((\psi, x, \alpha), \text{st}), \beta, \pi; \text{crs}) \neq 1$, output 0.
 - Else output $\text{Verify}^{(I)}(\psi, x, (\alpha, \beta, \gamma))$.

Proof. Completeness. This is immediate.

Knowledge Soundness.

Theorem 7.3. Let $\lambda \in \mathbb{N}$ be the security parameter. For every extractor $E^{(I)}$ for the interactive proof $(\text{Setup}, \text{Pre}^{(I)}, \text{Proof}^{(I)}, \text{Verify}^{(I)})$, there exists an interactive prover $\text{Proof}^{(I)} = (\text{Proof}_1^{(I)}, \text{Proof}_2^{(I)}, \text{Proof}_3^{(I)})$ and an extractor $\mathcal{E}$ for the non-interactive proof $(\text{Setup}, \text{Pre}, \text{Proof} = (\text{Proof}_1, \text{Proof}_2, \text{Proof}_3), \text{Verify})$ such that for all polynomial sized adversaries $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ making at most q many oracle queries, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\begin{aligned}
& \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{f, \text{Esk}}(\psi, x, (\alpha, \beta, \gamma, \text{st}, \pi)) = 1 \\ \wedge C(x, w) = 0 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{pk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, \text{st}_1) \leftarrow \mathcal{B}_1^{f, \text{Esk}}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}(h, C), \\ (x, \alpha, \beta, \gamma, \text{st}, \pi, \text{st}_2) \leftarrow \mathcal{B}_2^{f, \text{Esk}}(\text{vk}, \text{crs}, h, C, \text{st}_1), \\ \text{tr} := \text{all queries made to the oracle}, \\ w \leftarrow \mathcal{E}(C, x, \alpha, \beta, \gamma, \text{st}, \pi, \text{st}_2, \text{tr}) \end{array} \right] \\
& \leq t \cdot \Pr \left[\begin{array}{l} C \in \mathcal{C} \\ \wedge \text{Verify}^{(I)}(\psi, x, \alpha, \beta, \gamma) = 1 \\ \wedge C(x, w) = 0 \end{array} \middle| \begin{array}{l} (\text{crs}, \text{pk}, h, \text{td}, \text{sk}) \leftarrow \text{Setup}(1^\lambda), \\ (C, \text{state}_1) \leftarrow \text{Proof}_1^{(I)}(\text{crs}, \text{td}, h), \\ \psi \leftarrow \text{Pre}^{(I)}(h, C), \\ (x, \alpha, \text{state}_2) \leftarrow \text{Proof}_2^{(I)}(\text{vk}, \text{crs}, h, C, \text{state}_1), \\ \text{tr} := \text{all queries made to the oracle so far}, \\ w \leftarrow E^{(I)}(C, x, \alpha, \text{state}_2, \text{tr}), \\ \beta \leftarrow_{\$} \{0, 1\}^m \\ \gamma \leftarrow \text{Proof}_3^{(I)}(C, x, \alpha, \beta, \text{state}_2). \end{array} \right] + \text{negl}(\lambda)
\end{aligned}$$

The proof of this theorem is almost identical to that of Theorem 5.2, and in fact even more straightforward. It follows the blueprint of the proof of the standard Fiat-Shamir transformation with a random oracle, except that we will use the simulatability of the VRF to switch from uniform random challenge to the VRF evaluation. Note that this switch will be computational in nature as opposed to the statistical switch made possible by a random oracle. At a very high level, the proof follows the following template:

- In the first hybrid, the prover will first sample the challenge locally at random, rather than using the challenge sent by the interactive verifier. This swap will remain undetected by $\mathcal{B}$.
- In the next hybrid, we will use the simulatability of f to switch from the simulated to the real world.
- Next, we will add the additional check regarding the validity of the VRF proof that the non-interactive verifier needs to perform. This will only decrease the success probability.
- The rest of the proof details will have an identical structure as that of Theorem 5.2.

7.1 Application to Recursive Proofs

Recursive proofs in various forms are very sought after primitives and are one of the most important applications of proof systems in the real-world. Unfortunately, almost every such construction is heuristic. The reason for this is a fundamental disadvantage of the random oracle model – the verifier needs to query a random oracle. To see why this causes an issue, observe that, in the recursive step, the statement for which we are generating a proof is a function of the verifier circuit. Since, the verifier makes oracle queries, there is no way to even efficiently represent that as a circuit. The way around is to heuristically instantiate the random oracle with a cryptographic hash function and then perform the recursive step. While this has worked well for most applications, the recent work of [KRS25] exploits this very structure. The result below gets rid of heuristics by constructing recursive proofs where the verifier *does not* make any oracle queries and only needs to perform local operation.

We formally define the notion of a “two-step” recursive non-interactive argument of knowledge. This is a simplified notion for ease of exposition and can be naturally generalized to multi-leveled recursion.

Let C be some circuit and let us say that w is a valid witness to x if $C(x, w) = 1$. Let $(\text{Setup}_1, \text{Pre}_1, P_1, V_1)$ be a (pre-processing) SNARK. We define two NP languages $\mathcal{L}, \mathcal{L}'$ as

- $\mathcal{L} = \{(C, x) : \exists w \text{ such that } C(x, w) = 1\}$, and
- $\mathcal{L}' = \{(C, x, \text{crs}) : \exists \pi \text{ such that } V_1(\text{Pre}(\text{crs}, C), x, \pi) = 1\}$, where $\pi \leftarrow \text{Proof}_1(\text{crs}, C, x, w)$, and $\text{crs} \leftarrow \text{Setup}_1(1^\lambda)$.

Definition 7.4 (Recursive proof of knowledge). *A two-level recursive non-interactive proof for the language $\mathcal{L}$ consists of a pair tuples $(\text{Setup}_1, \text{Pre}_1, P_1, V_1)$ and $(\text{Setup}_2, \text{Pre}_2, P_2, V_2)$ such that*

- *The verifier circuit Verify_1 has an efficient representation as an arithmetic circuit,*
- *$(\text{Setup}_1, \text{Pre}_1, P_1, V_1)$ is a non-interactive argument for $\mathcal{L}$, and*
- *$(\text{Setup}_2, \text{Pre}_2, P_2, V_2)$ is a non-interactive argument for $\mathcal{L}'$.*

We extend this definition to the simulatable VRF oracle model.

Definition 7.5 (Recursive proof of knowledge in the simulatable VRF oracle model). *Let $(f.\text{Setup}, f.\text{Gen}, f.E, f.V)$ be an sVRF satisfying all properties of Definition 3.4. A two-level recursive non-interactive proof for the language $\mathcal{L}$ in the sVRF oracle model consists of a pair tuples $(\text{Setup}_1, \text{Pre}_1, P_1^{f.E}, V_1^{f.E})$ and $(\text{Setup}_2, \text{Pre}_2, P_2^{f.E}, V_2^{f.E})$ such that*

- *The verifier circuit $V_1^{f.E}$ has an efficient representation as an arithmetic circuit,*
- *$(\text{Setup}_1, \text{Pre}_1, P_1^{f.E}, V_1^{f.E})$ is a non-interactive argument with knowledge soundness for $\mathcal{L}$ in the sVRF oracle model, and*
- *$(\text{Setup}_2, \text{Pre}_2, P_2^{f.E}, V_2^{f.E})$ is a non-interactive argument with knowledge soundness for $\mathcal{L}'$ in the sVRF oracle model.*

The following is the main theorem.

Theorem 7.6. *Let*

- *$(f.\text{Setup}, f.\text{Gen}, f.E, f.V)$ be an sVRF satisfying all properties of Definition 3.4, and*
- *$(\text{Setup}, \text{Pre}, \text{Proof}, \text{Verify})$ be a non-interactive argument that satisfies adaptive round-by-round knowledge soundness in the simulatable VRF oracle model as per Construction 2.*

Let $(\text{Setup}', \text{Pre}', \text{Proof}', \text{Verify}')$ be a tuple of algorithms as described in Construction 3.

Then $(\text{Setup}, \text{Pre}, \text{Proof}, \text{Verify})$ and $(\text{Setup}', \text{Pre}', \text{Proof}', \text{Verify}')$ form a two-level recursive non-interactive proof for the language $\mathcal{L}$ in the sVRF oracle model.

For any circuit, input, witness triplet (C, x, w) , let

- $(\text{crs}, \text{vk}, h) \leftarrow \text{Setup}(1^\lambda)$,
- $\psi \leftarrow \text{Pre}(\text{crs}, \text{vk}, h, C)$,
- $\pi \leftarrow \text{Proof}^{f.E}(\text{crs}, \text{vk}, h, C, \psi, x, w)$.

Define a circuit C' that is hardcoded with ψ , and takes as input x and proof π and does the following:

$$C'_{[\psi]}(x, \pi) = 1 \iff \text{Verify}(\psi, x, \pi) = 1.$$

Construction 3 (Recursive Proof Construction).

- $\text{Setup}'(1^\lambda)$:
 - $\text{Output } (\text{crs}', \text{vk}', h') \leftarrow \text{Setup}(1^\lambda)$.
- $\text{Pre}'(\text{crs}', \text{vk}', h', C')$:
 - $\psi' \leftarrow \text{Pre}(\text{crs}', \text{vk}', h', C')$.
- $\text{Proof}'^{f.E_{sk}}(\text{crs}', \text{vk}', h', C', \psi', x, \pi)$:
 - $\Pi' := (\alpha', \beta', \gamma', \text{st}'', \pi') \leftarrow \text{Proof}^{f.E_{sk}}(\text{crs}', \text{vk}', h', (C', \psi', x), \pi)$.
- $\text{Verify}'^{f.E_{sk}}(\text{crs}', \text{vk}', h', \psi', x, \Pi')$:
 - $\text{Output } \text{Verify}^{f.E_{sk}}(\text{crs}', \text{vk}', h', \psi', x, \Pi')$.

Proof Outline. The proof of completeness is straightforward. The knowledge soundness of $(\text{Setup}', \text{Pre}', \text{Proof}', \text{Verify}')$ follows from a reduction to the knowledge soundness of $(\text{Setup}, \text{Pre}, \text{Proof}, \text{Verify})$. Essentially the extractor for $(\text{Setup}', \text{Pre}', \text{Proof}', \text{Verify}')$ would act as the malicious prover for $(\text{Setup}, \text{Pre}, \text{Proof}, \text{Verify})$ because it can output a valid witness, i.e., π , in this case.

Efficient representation: In order to explicitly define the circuit $C'_{[\psi]}$, it is crucial that the verification circuit $\text{Verify}^{f.E_{sk}}$ has an efficient representation. Observe that in Construction 2, the verification algorithm is *local* and *does not* make any oracle queries to $f.E_{sk}$. Therefore, it has an efficient representation as an arithmetic circuit.

The novelty of the above observation is that, in the sVRF oracle model, the verifier circuit only performs local operations, and therefore allows to construct provably secure recursive proofs. We now show that our techniques can be applied to state-of-the-art recursive proof system with minimum modifications.

7.2 Provably instantiating a state-of-the-art construction for recursive argument

Since we prove the soundness of the FS transform in the simulatable VRF model when the underlying interactive protocol is a Sigma proof, one natural place where our construction recipe can be used is recursive proofs where the base proof is a succinct non-interactive argument in the ROM via FS.

One such example is NOVA [KST22]- a state-of-the-art folding scheme based construction of a recursive succinct proof, in particular, an Incremental Verifiable Computation. We will discuss how we can instantiate NOVA in the simulatable VRF oracle model to guarantee provable security. This in turn gives us an explicit construction of a provably secure IVC in the sVRF model.

We write their construction of a three-round public-coin interactive folding scheme below and then present the non-interactive scheme in the sVRF oracle model. We have included background from [KST22] about folding schemes, R1CS and IVC in Section A in the Appendix for completeness. We skip the details in the main body to avoid redundancy.

At a high level, a folding scheme for a relation R is a protocol that reduces the task of checking two instances in R to the task of checking a single instance in R . It involves a prover and verifier that have two instances in R . In addition, the prover also knows the corresponding witnesses for each instance. Then, the two parties participate in an interactive protocol, at the end of which, the prover learns an instance-witness pair (u, w) and the verifier learns u such that $(u, w) \in R$.

Construction 4 (A Folding Scheme for Committed Relaxed R1CS). Consider a finite field $\mathbb{F}$ and a succinct, hiding, and homomorphic commitment scheme Com over $\mathbb{F}$. We define the generator and the encoder as follows.

- $\mathcal{G}(1^\lambda) \rightarrow \text{pp}$: output size bounds $m, n, \ell \in \mathbb{N}$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively.
- $\mathcal{K}(\text{pp}, (A, B, C)) \rightarrow (\text{pk}, \text{vk})$: output $\text{pk} \leftarrow (\text{pp}, (A, B, C))$ and $\text{vk} \leftarrow \perp$.

The verifier $\mathcal{V}$ takes two committed relaxed RICS instances $(\overline{E}_1, u_1, \overline{W}_1, x_1)$ and $(\overline{E}_2, u_2, \overline{W}_2, x_2)$. The prover $\mathcal{P}$, in addition to the two instances, takes witnesses to both instances, $(E_1, r_{E1}, W_1, r_{W1})$ and $(E_2, r_{E2}, W_2, r_{W2})$. Let $Z_1 = (W_1, x_1, u_1)$ and $Z_2 = (W_2, x_2, u_2)$. The prover and the verifier proceed as follows.

1. $\mathcal{P}$: Send $T := \text{Com}(\text{pp}_E, T, r_T)$, where $r_T \leftarrow_R \mathbb{F}$ and with cross term

$$T = AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1.$$

2. $\mathcal{V}$: Sample and send challenge $r \leftarrow_R \mathbb{F}$.
3. $\mathcal{V}, \mathcal{P}$: Output the folded instance $(\overline{E}, u, \overline{W}, x)$ where

$$\begin{aligned}\overline{E} &\leftarrow \overline{E}_1 + r \cdot T + r^2 \cdot \overline{E}_2, \\ u &\leftarrow u_1 + r \cdot u_2, \\ \overline{W} &\leftarrow \overline{W}_1 + r \cdot \overline{W}_2, \\ x &\leftarrow x_1 + r \cdot x_2.\end{aligned}$$

4. $\mathcal{P}$: Output the folded witness (E, r_E, W, r_W) , where

$$\begin{aligned}E &\leftarrow E_1 + r \cdot T + r^2 \cdot E_2, \\ r_E &\leftarrow r_{E1} + r \cdot r_T + r^2 \cdot r_{E2}, \\ W &\leftarrow W_1 + r \cdot W_2, \\ r_W &\leftarrow r_{W1} + r \cdot r_{W2}.\end{aligned}$$

Construction 5 (A Non-Interactive Folding Scheme). We construct non-interactive folding scheme in the simulatable VRF oracle model. Let $f = (f.\text{Setup}, f.\text{Gen}, f.\text{E}, f.\text{V})$ denote an sVRF. Let $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ represent our interactive folding scheme (Construction 4). We construct a non-interactive folding scheme (G, K, P, V) as follows:

- $G(1^\lambda)$: $\text{crs} \leftarrow f.\text{Setup}(1^\lambda)$, $(f.\text{ek}, f.\text{vk}) \leftarrow f.\text{Gen}(\text{crs})$ $p \leftarrow \mathcal{G}(1^\lambda)$, output $\text{pp} := (p, \text{crs}, f.\text{vk})$.
- $K(\text{pp}, (A, B, C))$: $\text{vk} \leftarrow \rho(\text{pp}, s)$ and $\text{pk} \leftarrow (\text{pp}, (A, B, C), \text{vk})$; output (vk, pk) .
- $P^{f.\text{E}}(\text{pk}, (u_1, w_1), (u_2, w_2))$: runs $\mathcal{P}((\text{pk}, \text{pp}, \text{pk}.(A, B, C)))$ to retrieve its first message $\overline{T}$, and sends $\overline{T}$ to $\mathcal{V}$; computes $(r, \pi) \leftarrow f.\text{E}(\text{crs}, f.\text{ek}, u_1, u_2, \overline{T})$, forwards this to $\mathcal{P}$, and outputs the resulting output along with (r, π) .
- $V(\text{vk}, u_1, u_2, \overline{T})$: First verify if $f.\text{V}(\text{crs}, \text{vk}, (u_1, u_2, \overline{T}), r, \pi) = 1$, else reject the proof; runs $\mathcal{V}$ with $\overline{T}$ as the message from the prover and with randomness r , and outputs the resulting output.

Lemma 7.7. *The construction is a non-interactive folding scheme with completeness and knowledge soundness.*

This lemma follows from the properties of the underlying interactive folding scheme and f in an identical manner as shown in the proof of Theorem 7.2.

We will not elaborate how to use this NIFS to construct an IVC. This step is identical as that of [KST22]. We therefore, provide a high level description for context and refer readers to their work for a detailed description.

Converting NIFS to IVC [KST22] - an overview. Recall that an IVC scheme allows a prover to show that $z_n = F^{(n)}(z_0)$ for some count n , initial input z_0 , and output z_n . [KST22] show how to construct an IVC scheme for any NP circuit F using NIFS for committed relaxed R1CS in Construction 5. In the IVC, the prover uses an augmented circuit F' , which, in addition to invoking F , performs additional local operations.

Intuitively, F' works as follows^[16]. F' takes as input two committed relaxed R1CS instances u_i and U_i . Suppose that U_i represents the correct execution of invocations $1, \dots, i-1$ of F' so long as u_i represents the correct execution of invocation i of F' . F' performs two tasks. First, it executes a step of the incremental computation: instance u_i contains z_i , which F' uses to output $z_{i+1} = F(z_i)$. Second, F' invokes the verifier of the non-interactive folding scheme to fold the task of checking u_i and U_i into the task of checking a single instance U_{i+1} . The IVC prover then computes a new instance u_{i+1} which attests to the correct execution of invocation $i+1$ of F' , thereby attesting that $z_{i+1} = F(z_i)$ and U_{i+1} is the result of folding u_i and U_i . This ensures that that U_{i+1} represents the correct execution of invocations $1, \dots, i$ of F' so long as u_{i+1} represents the correct execution of invocation $i+1$ of F' .

Let $(u_\perp, w_\perp)$ be the trivially satisfying instance–witness pair, where E, W , and x are appropriately sized zero vectors, $r_E = 0$, $r_W = 0$, and $\overline{E}$ and $\overline{W}$ are commitments of E and W , respectively.

Now, in iteration $i+1$, the IVC prover runs F' and computes u_{i+1} and U_{i+1} as well as the corresponding witnesses w_{i+1} and W_{i+1} . Because u_{i+1} and U_{i+1} together attest to the correctness of $i+1$ invocations of F' (which indirectly attests to $i+1$ invocations of F), the IVC proof Π_{i+1} is

$$\Pi_{i+1} = ((U_{i+1}, W_{i+1}), (u_{i+1}, w_{i+1})).$$

Therefore, we see that F' that is a part of the IVC circuit runs the underlying NIFS verifier during its execution. Since our construction of NIFS (Construction 5) ensures that the verifier does not need oracle access, this implies that the NIFS verifier and therefore the augmented circuit F' can be efficiently represented as an arithmetic circuit without resorting to heuristic instantiation of the random oracle, unlike what is necessitated in [KST22].

Lemma 7.8. (Informal (Section 5 in [KST22])) *Assuming the existence of a non-interactive folding scheme where the verification function can be efficiently represented as an arithmetic circuit, there exists a black-box construction of an IVC.*

8 Implementation and Evaluation

We implement the strongly simulatable VRF from [BCKL09], and use it to instantiate the Fiat-Shamir challenges in the GKR protocol. This VRF construction makes use of an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p , for some large prime (e.g. 255 bits), and we use the popular BLS12-381 curve.

Implementation Details. Our implementation is in pure Rust (approximately 3000 lines of code), based on an open-source implementation [web24] of the GKR protocol (without some of the more recent optimizations presented in [Tha13]). The VRF evaluation and its associated NIZK uses Groth-Sahai proofs, all of which is implemented using the BLS-12381 curve, for which we rely on the `arkworks` libraries [ac22]. The VRF secret key is sampled as $s \leftarrow \mathbb{Z}_p$, and its public key is computed as $\text{pk} \leftarrow h^s \in \mathbb{G}_2$, where $h \in \mathbb{G}_2$

^[16]The formal construction is slightly different and requires careful handling of some more details using a collision resistant hash function. See Section 5.1 in [KST22]

is a generator that is part of the public parameters. The input to the VRF is the GKR transcript, which must be mapped to $\mathbb{Z}_p$ using a collision-resistant hash-to-field – we use SHA2-256, again from the `arkworks` library, expanded to 64 bytes and then reduced mod p .

Experimental Setup. We deploy the VRF evaluator as a remote service backed by a Cloudflare worker [clo25]. Ideally, it would be implemented as a threshold cryptosystem^[17] to decentralize trust in any single server. However, we opt for simplicity in the experimental setup, and contend that threshold service can be thought of as adding additional latency to each VRF call. The prover and verifier are run on a Macbook Pro M4 laptop with 24 GB RAM. The round-trip latency between the local prover and remote VRF service was measured to be roughly 120 milliseconds.

Evaluation. We wish to study the overhead introduced by our VRF in the context of the GKR protocol. Since the hash-to-field operation is extremely efficient on modern CPUs, the dominant cost is in the VRF evaluation, especially considering the network-roundtrip required for the remote procedure call. Considering this, we steer away from evaluating our modified GKR protocol for different application circuits. Instead, we focus our attention on the specific case of inner-product computation $\langle a, b \rangle \doteq \sum_i^n a_i b_i$ between two vectors a and b of length n . In our experiments, we vary $n \in \{2^8, 2^{12}, 2^{16}\}$.

Vector Length	VRF-based GKR Prover Time		Hash-based GKR Proving Time
	local	remote	
2^8	5 ms	1.09 s	2.3 ms
2^{12}	40.9 ms	1.61 s	37 ms
2^{16}	595.3 ms	2.65 s	589 ms

Figure 1: Prover time for inner product computation, for varying vector sizes. We compare our construction (called VRF-based GKR) with the vanilla GKR based on SHA256 hash function for instantiating Fiat-Shamir (called Hash-based GKR). We measure our prover in two settings: 1) the VRF service is running locally (i.e., with zero network latency); and, 2) the VRF service running remotely as a Cloudflare worker.

Vector Length	Verifier Time		Proof Size Overhead	
	VRF-based	Hash-based	VRF-based	Hash-based
2^8	12.3 ms	0.6 ms	432 bytes	288 bytes
2^{12}	17.8 ms	0.9 ms	624 bytes	416 bytes
2^{16}	23.3 ms	1.2 ms	816 bytes	544 bytes

Figure 2: Verifier time and proof size overhead (for Random Oracle calls in addition to the GKR-specific proof elements) for inner product computation, for varying vector sizes. We compare our construction (called VRF-based GKR) with the vanilla GKR based on SHA256 hash function for instantiating Fiat-Shamir (called Hash-based GKR).

Despite the efficiency of the VRF construction, we find a significant overhead in the total running time due to the network round-trip of invoking the remote service. However, since the prover is idle during the waiting time, it is nevertheless efficient, considering the energy cost of generating a proof. Moreover, other proof systems which make fewer invocations of the random oracle compared to GKR will observe a smaller overhead, making our VRF-based compilation feasible for a variety of proof systems in practical applications.

Acknowledgements

Riddhi would like to thank Amit Sahai and Stefano Tessaro for useful discussions.

^[17]The VRF evaluation requires an inversion in the exponent, which can be implemented via a 1-round MPC protocol [SA19] (implementing a multiplication in the exponent) assuming a setup phase that generates a linear secret sharing of the VRF secret key.

9 References

- [ac22] arkworks contributors. `arkworks` zksnark ecosystem, 2022.
- [AY25] Gal Arnon and Eylon Yogev. Towards a white-box secure fiat-shamir transformation. In *Annual International Cryptology Conference*, pages 27–56. Springer, 2025.
- [Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, FOCS '01*, pages 106–115. IEEE Computer Society, 2001.
- [BBH⁺19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of kilian-based SNARGs. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 522–551, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.
- [BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.
- [BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the ROM do not exist. Cryptology ePrint Archive, Report 2024/728, 2024. <https://eprint.iacr.org/2024/728>.
- [BCKL09] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact e-cash and simulatable vrf's revisited. In *Proceedings of the 3rd International Conference Palo Alto on Pairing-Based Cryptography, Pairing '09*, page 114–131, Berlin, Heidelberg, 2009. Springer-Verlag.
- [BCL⁺21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.
- [BCMS20a] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. *Cryptology ePrint Archive*, 2020.
- [BCMS20b] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.
- [BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
- [BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In Tal Malkin and Chris Peikert, editors, *Advances*

- in *Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 649–680, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.
- [BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In *Advances in Cryptology - CRYPTO 2020*, pages 738–767. Springer, 2020.
 - [BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO’93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.
 - [CCG⁺23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O’Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In *Advances in Cryptology - EUROCRYPT 2023*, pages 379–404. Springer, 2023.
 - [CCH⁺19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-shamir: from practice to theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1082–1090. ACM, 2019.
 - [CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In *Theory of Cryptography - 13th International Conference, TCC 2016-A*, pages 389–415. Springer, 2016.
 - [CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-shamir and correlation intractability from strong KDM-secure encryption. In *Advances in Cryptology - EUROCRYPT 2018*, pages 91–122. Springer, 2018.
 - [CCS22] Megan Chen, Alessandro Chiesa, and Nicholas Spooner. On succinct non-interactive arguments in relativized worlds. In *Advances in Cryptology - EUROCRYPT 2022*, pages 336–366. Springer, 2022.
 - [CDG⁺25] Megan Chen, Pousali Dey, Chaya Ganesh, Pratyay Mukherjee, Pratik Sarkar, and Swagata Sasmal. Universally composable non-interactive zero-knowledge from sigma protocols via a new straight-line compiler. In *Public-Key Cryptography - PKC 2025 - 28th IACR International Conference on Practice and Theory of Public-Key Cryptography, Røros, Norway, May 12-15, 2025, Proceedings, Part I*, volume 15674 of *Lecture Notes in Computer Science*, pages 381–417. Springer, 2025.
 - [CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.
 - [CGSY24] Alessandro Chiesa, Ziyi Guan, Shahar Samocha, and Eylon Yogev. Security bounds for proof-carrying data from straightline extractors. In *Theory of Cryptography - 22nd International Conference, TCC 2024*, pages 464–496. Springer, 2024.
 - [cha25] Get a random number with vrf v2. <https://docs.chain.link/vrf/v2/subscription/examples/get-a-random-number>, 2025. Chainlink Documentation; accessed Sep. 15, 2025.

- [CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 68–79. IEEE, 2021.
- [CL07] Melissa Chase and Anna Lysyanskaya. Simulatable VRFs with applications to multi-theorem NIZK. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 303–322, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.
- [CLKL22] Thomas Chen, Hui Lu, Teeramet Kunpittaya, and Alan Luo. A review of zk-snarks. *arXiv preprint arXiv:2202.06877*, 2022.
- [clo25] 2025.
- [Con23] ConsenSys. Linea Bridge Smart Contract. <https://github.com/ConsenSys/linea-contracts>, 2023.
- [CSW22] Ran Canetti, Pratik Sarkar, and Xiao Wang. Triply adaptive UC NIZK. In *Advances in Cryptology – ASIACRYPT 2022, Part II*, *Lecture Notes in Computer Science*, pages 466–495. Springer, Heidelberg, Germany, December 7–11, 2022.
- [CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science, ICS ’10*, pages 310–331, 2010.
- [CY21] Alessandro Chiesa and Eylon Yogev. Subquadratic SNARGs in the random oracle model. In *Proceedings of the 41st Annual International Cryptology Conference, CRYPTO ’21*, pages 711–741. Springer, 2021.
- [CY24] Alessandro Chiesa and Eylon Yogev. Building cryptographic proofs from hash functions, 2024.
- [DJJ⁺25] Pratish Datta, Abhishek Jain, Zhengzhong Jin, Alexis Korb, Surya Mathialagan, and Amit Sahai. Incrementally verifiable computation for np from standard assumptions. In *Annual International Cryptology Conference*, pages 611–642. Springer, 2025.
- [DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO’92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.
- [DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Heidelberg, Germany.
- [FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of the 6th Annual International Cryptology Conference, CRYPTO ’86*, pages 186–194. Springer-Verlag, 1986.

- [GHK⁺25] Jacob Gorman, Lucjan Hanzlik, Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Pratik Sarkar, and Sri Aravinda Krishnan Thyagarajan. Vraas: Verifiable randomness as a service on blockchains. In *38th IEEE Computer Security Foundations Symposium, CSF 2025, Santa Cruz, CA, USA, June 16-20, 2025*, pages 331–346. IEEE, 2025.
- [GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS '03*, pages 102–113. IEEE Computer Society, 2003.
- [GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, 2015.
- [GLOW21] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 88–102. IEEE, 2021.
- [GMN22] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical snark aggregation. In *International Conference on Financial Cryptography and Data Security*, pages 203–229. Springer, 2022.
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.
- [Gro21] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. <https://eprint.iacr.org/2021/339>.
- [GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.
- [Hal] halo2 - The halo2 Book — zcash.github.io. <https://zcash.github.io/halo2/index.html>. [Accessed 04-07-2025].
- [HAN23] Mathias Hall-Andersen and Jesper Buus Nielsen. On valiant’s conjecture: impossibility of incrementally verifiable computation from random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 438–469. Springer, 2023.
- [Har23] Harmony. Harmony vrf. <https://docs.harmony.one/home/developers/tools/harmony-vrf>, 2023. Accessed: 2025-08-06.
- [HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In *Advances in Cryptology - EUROCRYPT 2022*, pages 520–549. Springer, 2022.
- [HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 850–858. IEEE Computer Society, 2018.

- [HLR21] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 750–760. ACM, 2021.
- [HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.
- [IAC26] IACR ePrint Archive. eprint report 2026/200. <https://eprint.iacr.org/2026/200.pdf>, 2026. Cryptology ePrint Archive, Report 2026/200; accessed February 2026.
- [JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In *Advances in Cryptology - EUROCRYPT 2021*, pages 3–32. Springer, 2021.
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC '92*, pages 723–732. ACM, 1992.
- [KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs and PPAD hardness from the decisional diffie-hellman assumption. In *Advances in Cryptology - EUROCRYPT 2023*, pages 470–498. Springer, 2023.
- [KMM⁺24] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive VSS using class groups and application to DKG. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 4286–4300. ACM, 2024.
- [KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of fiat-shamir for proofs. In *Advances in Cryptology - CRYPTO 2017*, pages 224–251. Springer, 2017.
- [KRS25] Dmitry Khovratovich, Ron D Rothblum, and Lev Soukhanov. How to prove false statements: Practical attacks on fiat-shamir. In *Annual International Cryptology Conference*, pages 3–26. Springer, 2025.
- [KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022.
- [KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.
- [Lab22] Chainlink Labs. Chainlink vrf: On-chain verifiable randomness. <https://blog.chainlink.com/chainlink-vrf-on-chain-verifiable-randomness/>, September 2022. Accessed: 2025-08-06.
- [Lab26] Succinct Labs. What is succinct? <https://docs.succinct.xyz/docs/protocol/what-is-succinct>, 2026. Succinct Protocol documentation; accessed February 2026.

- [MAG⁺24] Daniel Marin, Michel Abdalla, Paul Govereau, Jens Groth, Samuel Judson, Kristian Sosnin, Guru Vamsi Policharla, and Yinuo Zhang. Nexus 1.0: Enabling verifiable computation. Technical report, Nexus Labs, Jan 2024. Whitepaper; Nexus project foundational vision and zkVM details.
- [Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. Preliminary version appeared in FOCS ’94.
- [MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press.
- [NR22] Valeria Nikolaenko and Sam Ragsdale. On-chain trusted setup ceremony. <https://a16zcrypto.com/posts/article/on-chain-trusted-setup-ceremony/>, 2022. a16z Crypto, Sep. 7, 2022.
- [Ora23] Binance Oracle. Verifiable random function (vrf) overview. <https://oracle.binance.com/docs/vrf/overview/>, 2023. Accessed: 2025-08-06.
- [plo] GitHub - 0xPolygonZero/plonky2 — github.com. <https://github.com/0xPolygonZero/plonky2>. [Accessed 04-07-2025].
- [Pol24] Polyhedra Network. Expander, 2024. Accessed: 2025-01-15.
- [PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT’96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.
- [PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *Advances in Cryptology - CRYPTO 2019*, pages 89–114. Springer, 2019.
- [Raq25] ”35+ blockchain rng use cases enabled by chainlink vrf”, 2025. <https://chain.link/education-hub/rng-in-blockchain-use-cases>.
- [Res23] Supra Research. Supra vrf service. Technical report, Supra, 2023. Technical whitepaper on distributed verifiable random function service.
- [SA19] Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. Cryptology ePrint Archive, Paper 2019/768, 2019.
- [T⁺22] Justin Thaler et al. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022.
- [Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 71–89, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [TX24] Alin Tomescu and Zhuolun Xiang. Roll with move: Secure, instant randomness on aptos. Blog post, Aptos Labs, June 9 2024. Accessed Sep. 15, 2025.

- [Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.
- [web24] 2024.
- [WTS⁺18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *Proceedings of the 39th IEEE Symposium on Security and Privacy, S&P '18*, pages 926–943. IEEE Computer Society, 2018.
- [XZZ⁺19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Proceedings of the 39th Annual International Cryptology Conference, CRYPTO '19*, pages 733–764. Springer, 2019.
- [ZGK⁺17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proceedings of the 38th IEEE Symposium on Security and Privacy, S&P '17*, pages 863–880. IEEE Computer Society, 2017.
- [ZLW⁺21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 159–177. ACM, 2021.

A Provably instantiating NOVA: Prelims

We begin by briefly describing folding schemes and the construction of the folding scheme as described in [KST22].

Intuitively, a folding scheme for a relation R is a protocol that reduces the task of checking two instances in R to the task of checking a single instance in R .

Definition A.1 (Folding Scheme). *Consider a relation R over public parameters, structure, instance, and witness tuples. A folding scheme for R consists of a PPT generator algorithm $\mathcal{G}$, a deterministic decoder algorithm $\mathcal{K}$, and a pair of PPT algorithms $\mathcal{P}$ and $\mathcal{V}$ denoting the prover and verifier respectively, with the following interface:*

- $\mathcal{G}(1^\lambda) \rightarrow \text{pp}$: On input security parameter λ , sample public parameters pp .
- $\mathcal{K}(\text{pp}, s) \rightarrow (\text{pk}, \text{vk})$: On input pp and common structure s between instances to be folded, outputs a prover key pk and a verifier key vk .
- $\mathcal{P}(\text{pk}, (u_1, w_1), (u_2, w_2)) \rightarrow (u, w)$: On input instance–witness tuples (u_1, w_1) and (u_2, w_2) , outputs a new instance–witness tuple (u, w) of the same size.
- $\mathcal{V}(\text{vk}, u_1, u_2) \rightarrow u$: On input instances u_1 and u_2 , outputs a new instance u .

Let

$$(u, w) \leftarrow \mathcal{P}(\text{pk}, u_1, w_1, u_2, w_2), \quad u \leftarrow \mathcal{V}(\text{vk})(u_1, u_2)$$

denote the verifier's output instance u and the prover's output witness w from the interaction of $\mathcal{P}$ and $\mathcal{V}$ on witnesses $(u_1, w_1), (u_2, w_2)$, prover key $\mathbf{pk}$, verifier key $\mathbf{vk}$ and instances (u_1, u_2) . Likewise, let

$$\mathbf{tr} = \langle \mathcal{P}(\mathbf{pk}, u_1, w_1, u_2, w_2), \mathcal{V}(\mathbf{vk})(u_1, u_2) \rangle$$

denote the corresponding interaction transcript.

A folding scheme satisfies perfect completeness if for all PPT adversaries $\mathcal{A}$

$$\Pr \left[\begin{array}{l} \mathbf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, u_1, w_1, u_2, w_2) \leftarrow \mathcal{A}(\mathbf{pp}), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{pp}, s), \\ (u, w) \leftarrow \mathcal{P}(\mathbf{pk}, u_1, w_1, u_2, w_2), \\ u' \leftarrow \mathcal{V}(\mathbf{vk})(u_1, u_2) \end{array} : (u, w) \in R \right] = 1.$$

A folding scheme satisfies knowledge soundness if for any expected polynomial-time adversary $\mathcal{P}^*$ there is an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr \left[\begin{array}{l} \mathbf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, u_1, u_2) \leftarrow \mathcal{P}^*(\mathbf{pp}), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{pp}, s), \\ \mathbf{tr} \leftarrow \langle \mathcal{P}^*(\mathbf{pp}), \mathcal{V}(\mathbf{vk})(u_1, u_2) \rangle, \\ (u, w) \leftarrow \mathcal{E}(\mathbf{pp}, \mathbf{tr}) \end{array} : (u, w) \in R \right] = \text{negl}(\lambda)$$

where ρ denotes arbitrary input randomness for $\mathcal{P}^*$. We call a transcript $\mathbf{tr}$ an accepting transcript if $\mathcal{P}^*$ outputs folded witnesses w for the folded instance u . We consider a folding scheme non-trivial if the communication costs and $\mathcal{V}$'s computation are at most the size of u . Perfect completeness in the folding scheme checks that instances sent by $\mathcal{P}$ are valid, whereas knowledge soundness ensures that $\mathcal{V}$ checks witnesses output by $\mathcal{P}$ for each of the original instances.

Definition A.2 (Non-Interactive). A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. This single message is denoted as an output of $\mathcal{P}$, and an input to $\mathcal{V}$.

Definition A.3 (IVC). An incrementally verifiable computation (IVC) scheme is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$ denoting the generator, the prover, the verifier, and the encoder respectively. An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies perfect completeness if for any PPT adversary $\mathcal{A}$

$$\Pr \left[\begin{array}{l} \mathbf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ F(i, z_0, z_i, z_{i-1}, \omega_{i-1}, \Pi_{i-1}) \leftarrow \mathcal{A}(\mathbf{pp}), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{pp}, F), \\ z_i \leftarrow F(z_{i-1}, \omega_{i-1}), \\ \mathcal{V}(\mathbf{vk}, i-1, z_0, z_{i-1}, \Pi_{i-1}) = 1, \\ \Pi_i \leftarrow \mathcal{P}(\mathbf{pk}, i, z_0, z_i; z_{i-1}, \omega_{i-1}, \Pi_{i-1}) \end{array} : \mathcal{V}(\mathbf{vk}, i, z_0, z_i, \Pi_i) = 1 \right] = 1.$$

where F is a polynomial-time computable function. Likewise, an IVC scheme satisfies knowledge soundness if for any constant $n \in \mathbb{N}$, and for all expected polynomial-time adversaries $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr_r \left[\begin{array}{l} z_n = z \text{ where } z_{i+1} \leftarrow F(z_i, \omega_i) \\ \forall i \in \{0, \dots, n-1\} \\ (F(z_0, z), \Pi) \leftarrow \mathcal{P}^*(\mathbf{pp}, r), \\ (\omega_0, \dots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathbf{pp}, r) \end{array} \right] \approx \Pr_r \left[\begin{array}{l} \mathbf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (F(z_0, z), \Pi) \leftarrow \mathcal{P}^*(\mathbf{pp}, r), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{pp}, F) \end{array} : \mathcal{V}(\mathbf{vk}, (n, z_0, z), \Pi) = 1 \right],$$

where r denotes an arbitrarily long random tape.

An IVC scheme satisfies *succinctness* if the size of the IVC proof Π does not grow with the number of applications n .

Definition A.4 (R1CS). Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. An instance $x \in \mathbb{F}^\ell$ consists of public inputs and outputs and is satisfied by a witness $W \in \mathbb{F}^{m-\ell-1}$ if

$$(A \cdot Z) \circ (B \cdot Z) = C \cdot Z,$$

where $Z = (W, x, 1)$.

Definition A.5 (Relaxed R1CS). Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The relaxed R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix.

A relaxed R1CS instance consists of an error vector $E \in \mathbb{F}^m$, a scalar $u \in \mathbb{F}$, and public inputs and outputs $x \in \mathbb{F}^\ell$. An instance (E, u, x) is satisfied by a witness $W \in \mathbb{F}^{m-\ell-1}$ if

$$(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E,$$

where $Z = (W, x, u)$.

Definition A.6 (Committed Relaxed R1CS). Consider a finite field $\mathbb{F}$ and a commitment scheme Com over $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively.

The committed relaxed R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix.

A committed relaxed R1CS instance is a tuple $(\overline{E}, u, \overline{W}, x)$, where $\overline{E}$ and $\overline{W}$ are commitments, $u \in \mathbb{F}$, and $x \in \mathbb{F}^\ell$ are public inputs and outputs.

An instance $(\overline{E}, u, \overline{W}, x)$ is satisfied by a witness $(E, r_E, W, r_W) \in (\mathbb{F}^m, \mathbb{F}, \mathbb{F}^{m-\ell-1}, \mathbb{F})$ if

$$\overline{E} = \text{Com}(\text{pp}_E, E, r_E), \quad \overline{W} = \text{Com}(\text{pp}_W, W, r_W), \quad \text{and} \quad (A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E,$$

where $Z = (W, x, u)$.