Classical and Quantum Security of Elliptic Curve VRF, via Relative Indifferentiability

Chris Peikert*† Jiayu Xu*‡

February 18, 2023

Abstract

Verifiable random functions (VRFs) are essentially pseudorandom functions for which selected outputs can be proved correct and unique, without compromising the security of other outputs. VRFs have numerous applications across cryptography, and in particular they have recently been used to implement committee selection in the Algorand protocol.

Elliptic Curve VRF (ECVRF) is an elegant construction, originally due to Papadopoulos *et al.*, that is now under consideration by the Internet Research Task Force. Prior work proved that ECVRF possesses the main desired security properties of a VRF, under suitable assumptions. However, several recent versions of ECVRF include changes that make some of these proofs inapplicable. Moreover, the prior analysis holds only for *classical* attackers, in the random-oracle model (ROM); it says nothing about whether any of the desired properties hold against *quantum* attacks, in the quantumly accessible ROM. We note that certain important properties of ECVRF, like uniqueness, do *not* rely on assumptions that are known to be broken by quantum computers, so it is plausible that these properties could hold even in the quantum setting.

This work provides a multi-faceted security analysis of recent versions of ECVRF, in both the classical and quantum settings. First, we motivate and formally define new security properties for VRFs, like non-malleability and binding, and prove that recent versions of ECVRF satisfy them (under standard assumptions). Second, we identify a subtle obstruction in proving that recent versions of ECVRF have *uniqueness* via prior indifferentiability definitions and theorems, even in the classical setting. Third, we fill this gap by defining a stronger notion called *relative indifferentiability*, and extend prior work to show that a standard domain extender used in ECVRF satisfies this notion, in both the classical and quantum settings. This final contribution is of independent interest and we believe it should be applicable elsewhere.

1 Introduction

A Verifiable Random Function (VRF), as introduced by Micali, Rabin, and Vadhan [MRV99], is a cryptographic primitive that allows one to prove that outputs of a pseudorandom function (PRF) are correct, without compromising the pseudorandomness of other outputs. More precisely, a prover first generates a secret

*Algorand, Inc. Most of this work was done while at Algorand.

†Computer Science and Engineering, University of Michigan. This material is also supported by DARPA under Agreement No. HR00112020025. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

‡Electrical Engineering and Computer Science, Oregon State University.

key sk and a related public key pk. Then for any function input α, the prover can use sk to compute the function output $\beta := F_{sk}(\alpha)$, together with a proof $\pi \leftarrow \mathsf{Prove}_{sk}(\alpha)$ of its correctness. A verifier can then use pk to check a claimed proof for a given input-output pair. Importantly, for any fixed pk—even a maliciously generated one—each input should have a *unique* output for which it is feasible to prove correctness. Moreover, outputs for which proofs have not yet been published should remain *pseudorandom*. Uniqueness and pseudorandomness are just the two main security properties we ask of a VRF, and certain applications may require other properties (see below).

VRFs have found applications in, for example, zero-knowledge proofs [MR01], lottery systems [MR02], electronic cash [BCKL09], and DNS security [PWH+17, VGP+18]. Recently, they have received wide attention thanks to their applications to cryptocurrencies like Algorand [GHM+17, CM19], Cardano [BGK+18, DGKR18], and the Dfinity Blockchain [AMNR18]. More specifically, VRFs are used to implement *cryptographic sortition*, in which a small 'committee' of protocol participants is selected periodically; a party is in the committee when its VRF output (on a certain public input) is within some specified range. The VRF's uniqueness property helps to ensure that a party cannot improperly include itself in the committee, while the pseudorandomness property conceals the committee's makeup until the members verifiably reveal themselves.[1]

ECVRF. A particularly elegant and efficient VRF construction is the Elliptic Curve VRF (ECVRF) of Papadopoulos *et al.* [PWH+17], an 'Internet draft' version of which [GRPV22] is currently under consideration by the Crypto Forum Research Group of the Internet Research Task Force. Its security is analyzed in the Random Oracle Model (ROM), under the Decisional Diffie–Hellman (DDH) assumption for particular elliptic-curve groups. In practice, an implementation of ECVRF is used for cryptographic sortition in the deployed Algorand protocol.

There are certain differences between the original version of ECVRF [PWH+17] and some recent versions of the Internet draft [GRPV22], which make significant parts of the security analysis from [PWH+17] no longer applicable (see below for details). Moreover, the prior analysis is only for *classical* attackers, in the classical ROM; it says nothing about whether the desired security properties hold against a *quantum* attacker, including one in the *quantumly accessible* ROM (QROM) [BDF+11], where the adversary can query a random oracle on superpositions of inputs.

At first glance, it may appear nonsensical to consider the 'post-quantum' security of a cryptographic primitive like ECVRF that relies on 'pre-quantum' assumptions like the hardness of DDH or computing discrete logs, which do not hold in the quantum setting due to Shor's algorithm [Sho94]. However, a closer look reveals that while ECVRF certainly requires such an assumption for *pseudorandomness* (because given a discrete-log oracle, it is trivial to compute the secret key from the public key), it is less clear whether there are efficient quantum attacks on ECVRF's other desirable properties, like *uniqueness*.

Mixed pre- and post-quantum security. Specific motivation for understanding the mixture of ECVRF's pre- and post-quantum security properties comes from its use in applications like sortition. Here, pseudorandomness is needed only in the 'medium term', i.e., during the public key's lifetime in the protocol, to conceal which parties will be selected for committees. So, a quantum adversary that breaks pseudorandomness many years in the future, after a key is no longer in use, may not be a concern at all.

[1] We caution that while uniqueness is a critical property for secure sortition, it *alone* does not suffice to prevent a malicious party from improperly including itself in committees. Specifically, it does not preclude the generation of a malformed public key that induces a *constant* function (whose outputs are always in the relevant range). Sortition protocols include additional measures to ensure that even maliciously generated public keys do not result in biases like this.

By contrast, uniqueness may be needed in the 'long term': proofs of correct VRF evaluation may need to be verified far into the future, e.g., to ensure correct committee membership when verifying a blockchain's history. Without post-quantum uniqueness, a future quantum attacker could potentially forge valid-looking proofs and 'fork' the chain from any point in its history.

Therefore, systematically investigating post-quantum security is important for evaluating the actual consequences of quantum computers for ECVRF and its applications. Positive results may allow new versions of these applications to use simpler or less costly protections against future quantum attacks.

1.1 Contributions and Technical Overview

This work provides a multi-faceted security analysis of ECVRF as defined in (recent versions of) the Internet draft [GRPV22], in both the classical and quantum settings. The main contributions are threefold.

Non-malleability and binding. First, we propose a new security notion called *non-malleability* (Section 5.1), which essentially says that it is infeasible to generate a valid proof (for an honestly generated public key, but an adversarially chosen input and output) that is different from all the proofs generated by the honest prover. This property addresses the following potential issue in an application to distributed ledgers or cryptocurrencies: an honest prover may announce a valid VRF proof, but while the proof is being 'gossiped' through the network, a malicious gossiper might try to modify the proof to a different valid one. This may make it difficult for honest parties to reach consensus on which proof is the 'correct' one.[2]

In Section 5.3, we show that ECVRF (as defined in versions 10 and later of [GRPV22]) is non-malleable in the ROM, assuming the hardness of the discrete-logarithm problem.[3] (Conversely, since discrete logs are easy to compute in the quantum setting, ECVRF is easily malleable by a quantum attacker.) Our proof technique is similar to the one for Schnorr's signature scheme [Sch89], using the generic forking lemma [BN06], though the details are somewhat different. We note that this results in a quadratically loose concrete security bound (see Theorem 5.6). However, just as with Schnorr signatures, we do not know if there is a matching attack, i.e., the looseness might just be an artifact of the proof technique. (See Section 1.2 for further discussion.)

Additionally, in Section 5.2 we show that ECVRF satisfies another new notion we call *full binding* (Section 5.1), assuming only that the hash functions used in ECVRF are collision resistant. In particular, this proof holds even in the quantum setting. Full binding means that it is infeasible to generate two distinct public-key-input-output tuples along with a single proof that is valid for both of them. In other words, a valid proof is bound to a unique key, input, and output. (This notion is quite similar to binding concepts for signatures, as recently defined in [BCJZ21, CGN20, CDF+21].) Lastly on this front, we show in Theorem 5.4 that non-malleability combined with 'trusted' binding (a weaker notion than full binding) implies *strong non-malleability*, i.e., given oracle access to the prover, it is infeasible to generate a 'new' valid input-output-proof tuple.

Uniqueness: classical and post-quantum. Second, in Section 4 we prove the uniqueness of ECVRF *as defined in the Internet draft* [GRPV22], against both classical and quantum attacks in the (Q)ROM. To see that post-quantum uniqueness is even plausible, we first observe that the prior proof of classical uniqueness is *information theoretic*: it does not rely on any intractability assumption (e.g., the hardness of computing discrete logs), only the adversary's bounded query complexity in the ROM. This is because an ECVRF proof

[2] We stress that this is only a hypothetical scenario, and we do not know of any proposed protocol that actually has this issue. However, future applications might implicitly assume non-malleability of VRF proofs, for reasons like the ones described above.

[3] Version 10 was updated at our suggestion to achieve non-malleability; previous versions were trivially malleable.

is essentially a *statistically sound* interactive proof of discrete-log equality [CP92], made non-interactive via the Fiat–Shamir transform [FS86]. Indeed, we show that the soundness of this non-interactive proof system (against a classical or quantum attacker) implies the uniqueness of ECVRF (against the same kind of attacker). However, attempting to prove the soundness of ECVRF as defined in versions 2–10 of [GRPV22] ends up revealing significant and subtle difficulties.[4]

Although the differences between the original and later versions of the Internet draft are syntactically minor and well motivated, it turns out to be non-trivial to adapt the prior soundness proof to the latter. The key difference is that, in the original version, the 'challenge' in the proof is defined as $c := \mathsf{H}(X, \mathsf{HTC}(\alpha), W)$, whereas in [GRPV22, versions 2 through 10] it is defined as $c := \mathsf{H}(\mathsf{HTC}(X, \alpha), W)$, where H and HTC are modeled as random oracles (and X is the public key, α is the VRF input, and W consists of some additional data). Because X, which is part of the 'statement' to be proved, is no longer an explicit input to H, we can no longer directly apply known (classical or quantum) soundness theorems for the Fiat–Shamir transform [BR93, Unr17, DFMS19] to the modified construction.

At first glance, it may seem that the above issue can easily be overcome by using the fact that the 'domain extender'

$$\mathsf{C}(x_1, x_2) := \mathsf{H}_2(\mathsf{H}_1(x_1), x_2) \tag{1.1}$$

is *indifferentiable from a random oracle* [MRH04, CDMP05], even in the quantum setting [Zha19]. So, an adversary has essentially the same advantage in breaking ECVRF's soundness in the 'real' world as in the 'ideal' world, where the challenge is defined as $c := \mathsf{H}'(X, \alpha, W)$ for a random oracle H', and HTC, H are simulated using access to H'.

Unfortunately, this application of indifferentiability does not yield any useful conclusion for our purposes, because *it is easy to break soundness in the 'ideal' world*. The essence of the problem is that the existing indifferentiability definitions give the simulator too much power in our context. More specifically, the simulator is allowed to 'program' the value of $H := \mathsf{HTC}(X, \alpha)$, and the soundness experiment does not make this query until *after* the adversary sees the challenge $c = \mathsf{H}'(X, \alpha, W)$ and outputs its attempted break. Since H is part of the 'statement' that the adversary is attempting to prove, the simulator can easily tailor H based on c, to yield a false statement for which the adversary's proof verifies.

We stress that the above-described issue does not translate to an actual *attack* on any version of ECVRF; it merely shows that the prior indifferentiability definitions and theorems are unsuitable for proving soundness of certain versions of [GRPV22]. In particular, the simulator that is used to prove indifferentiability does not exhibit the above-described 'malicious' behavior, but this fact is not exposed by the definitions and theorems. To bridge this gap, we define and achieve an alternative notion called *relative indifferentiability* (summarized below), which circumvents the above-described difficulties by suitably weakening the simulator in the ideal world. Combining this with a careful sequence of steps, including the use of prior soundness theorems for Fiat–Shamir [BR93, DFMS19], we ultimately prove the soundness, and hence uniqueness, of ECVRF in the classical and quantum settings; see Theorem 4.14 for the formal statement.

We remark that in the classical setting, our ultimate concrete security bounds for uniqueness are fairly tight, and are even meaningful for typical ECVRF parameters (i.e., concrete elliptic-curve groups and challenge spaces). However, in the quantum setting the concrete bounds are necessarily looser, because they inherit the prior Fiat–Shamir and indifferentiability bounds, which are nearly matched by known quantum attacks. Therefore, ECVRF parameters will likely need to be adjusted in order to obtain meaningful levels of concrete quantum security.

[4] In response to our observations, version 11 of [GRPV22] introduced a change to restore a more straightforward proof of (classical) soundness using standard techniques. However, it is still useful to formally support the approach taken in earlier versions, which may be used elsewhere, and to investigate post-quantum security.

Relative indifferentiability and find-input oracles. As our final main contribution, in Section 3 we propose a stronger notion of indifferentiability called *indifferentiability relative to an auxiliary oracle*—or *relative indifferentiability* for short—and prove that the domain extender from Equation (1.1) satisfies this notion (for a suitable kind of auxiliary oracle) in both the classical and quantum settings. This contribution is of independent interest, and we believe that it should be applicable elsewhere.

Essentially, relative indifferentiability is an analog of ordinary indifferentiability where the construction (in the real world), the simulator (in the ideal world), and the distinguisher all have access to the same auxiliary oracle. Critically, the simulator has only *query access* to this auxiliary oracle; it does not get to simulate or 'program' it. For this reason, relative indifferentiability is a strengthening of ordinary indifferentiability, as long as the auxiliary oracle is (efficiently) computable.

Our main theorems on this front (Theorems 3.13 and 3.18) say that the domain extender from Equation (1.1) is indifferentiable from a random oracle, relative to a slightly augmented 'inner' function H_1 (or HTC in the ECVRF context), in both the classical and quantum settings. Essentially, making the inner function 'honest' by removing it from the simulator's control circumvents the above-described difficulties in proving soundness of ECVRF using indifferentiability.

Our relative-indifferentiability theorems for domain extension are analogs of prior ones showing ordinary indifferentiability [CDMP05, Zha19], and our proofs can be seen as 'refactorings' of the prior proofs. The key observation is that in the prior proofs, the simulators use very little of their ability to program the inner function: they merely simulate it 'honestly,' as a ('lazy' classical, or 'compressed' quantum) random oracle. So, the inner function can be 'moved outside of' the simulators, and instead be made an auxiliary oracle. However, the simulators also need to be able to *look up prior queries* (if any) to the inner function that yields certain outputs. We address this by augmenting the auxiliary oracle with an additional 'find-input' interface that exposes exactly this functionality.

Ultimately, in an application of relative indifferentiability (like ours), one would typically need to show that a construction is secure in the 'ideal' world, where the attacker has access to the auxiliary oracle, e.g., a find-input oracle. In many (but certainly not all!) cases, including our own, this is fairly straightforward, because the adversary already 'knows' all the queries that are made in the attack experiment (i.e., the experiment does not make any secret queries). This task is more subtle in the quantum setting, but one can use tools for 'recording' quantum queries, as provided in [Zha19].

1.2 Related and Future Work

As mentioned above, our non-malleability theorem for ECVRF has a quadratic concrete security loss. It is natural to ask two questions: first, is such a loss inherent for black-box reductions from the ordinary discrete logarithm problem? In the other direction, is there a tighter reduction under a stronger assumption, or in a stronger model? Given recent tighter security analysis for Schnorr signatures in the Algebraic Group Model (AGM) [FPS20] and under "higher-moment" discrete-log assumptions [RS21], analogous results for ECVRF's non-malleability seem plausible.

The recent work of [ESLR22] formalizes a folklore generic construction for VRFs, and analyzes the uniqueness of all VRF schemes that fit this framework, including ECVRF. However, its analysis is in the classical setting; in particular, it only considers the ROM, not the quantumly-accessible ROM. A future direction would be to extend the analysis to the QROM, which would cover the uniqueness of ECVRF as a special case.

Acknowledgments. We thank Mark Zhandry and Dominique Unruh for very helpful discussions about compressed oracles and our 'find-input' variation thereof, Leo Reyzin for helpful discussions about ECVRF

and its variants, and Iñigo Azurmendi, Peter Găzi, and Romain Pellerin for initial observations about the malleability of early versions of ECVRF.

2 Preliminaries

We write $x \leftarrow X$ for sampling an element x uniformly at random from a finite set X. For a randomized algorithm $\mathcal{A}$, we write $y := \mathcal{A}(x_1, \ldots; \rho)$ for running $\mathcal{A}$ on input $x_1, \ldots$ with random tape ρ to obtain output y, and we write $y \leftarrow \mathcal{A}(x_1, \ldots)$ when ρ is chosen uniformly at random. If $\mathcal{A}$ is deterministic, we write $y := \mathcal{A}(x_1, \ldots)$.

2.1 Oracles

A *quantumly accessible* oracle is an oracle that, when queried, applies some unitary U on particular register(s) of the querying algorithm. Any quantumly accessible oracle also has a generic *classical* interface, which additionally measures the query register(s) before and after applying U. In particular, this allows a classical algorithm to query the oracle (in a more limited way); note that here the query register(s), which hold classical values, are already 'measured' prior to the query.

We say that a procedure with access to an oracle has *query complexity* Q if it makes at most Q queries to that oracle. For a procedure with access to multiple oracles, its query complexity Q is a tuple whose ith component is an upper bound on the number of queries it makes to its ith oracle. For notational convenience, we sometimes also let Q denote the sum of its components, i.e., the procedure's total query complexity.

In the (classical) random-oracle model (ROM) [BR93], a uniformly random function H (having a specified finite domain and range) is chosen at the beginning of the security experiment, and all parties—including the 'honest' algorithms of the cryptographic construction, and the adversary attacking it—have classical query access to H as an oracle. The *quantum* random-oracle model (QROM) [BDF+11] is defined in the same way, except with quantum oracle access to H. Specifically, the oracle's unitary is defined as $U|x, y\rangle = |x, y \oplus H(y)\rangle$, where $\oplus$ denotes the group operation on the range of H (which is a group without loss of generality).

2.2 Cryptographic Assumptions

Here and in all subsequent definitions, there may be some fixed public parameters (e.g., the description of a group) that are known to all algorithms and not explicitly written.

Definition 2.1 (Discrete Logarithm Problem). Let $\mathbb{G}$ be a cyclic group with known order q and known generator B. We say that the discrete logarithm problem is (t, ϵ)-hard for $(\mathbb{G}, q, B)$ if, for any algorithm $\mathcal{A}$ running in time at most t,

$$\mathbf{Adv}^{\mathrm{DL}}(\mathcal{A}) := \Pr\left[\begin{array}{c} x \in \mathbb{Z}_q \text{ and} \\ X = B^x \end{array} \;:\; \begin{array}{c} X \leftarrow \mathbb{G} \setminus \{e\} \\ x \leftarrow \mathcal{A}(X) \end{array} \right] \leq \epsilon.$$

Note that the element X is chosen uniformly from the *non-identity* elements of the group $\mathbb{G}$. We define the discrete logarithm problem in this way so that it is identical to the problem of finding the secret key in the ECVRF construction (Algorithm 1), where the public key is likewise required to be a non-identity element. This is needed for certain 'full' security properties; see [GRPV22, Section 3].

Definition 2.2 (Collision Resistance). Let H be a function with domain D. We say that H is (t, ϵ)-collision resistant if, for any algorithm $\mathcal{A}$ running in time at most t,

$$\mathbf{Adv}^{\mathrm{CR}}(\mathcal{A}) := \Pr\left[\begin{array}{c} x_0, x_1 \in D \text{ and} \\ x_0 \neq x_1 \text{ and} \\ H(x_0) = H(x_1) \end{array} : (x_0, x_1) \leftarrow \mathcal{A}()\right] \leq \epsilon.$$

Note that in the above experiment, $\mathcal{A}$ takes no explicit input. However, in the random-oracle model, the oracle's outputs act as $\mathcal{A}$'s inputs, and the probability is quantified over the oracle and $\mathcal{A}$'s random choices.

In the standard model, the above notion is not meaningful as defined, since there *exists* an adversary that simply outputs a 'hard-coded' collision (whenever the function's range is smaller than its domain). This issue is usually addressed by considering a *keyed family* $\{H_k\}$ of functions, giving $\mathcal{A}$ a randomly chosen key k as input, requiring it to find a collision in H_k, and taking the probability over the choice of k and $\mathcal{A}$'s random tape. However, even without this change, it is still meaningful to consider the advantage of a *specific* adversary, e.g., a reduction that breaks collision resistance given oracle access to an adversary against some other security property. This is the approach we take in this work.

2.3 Verifiable Random Functions

Definition 2.3 (Verifiable Random Function). Let $\mathcal{X}, \mathcal{Y}$ respectively denote a domain and range, with $\mathcal{Y}$ finite. A *verifiable random function (VRF)* from $\mathcal{X}$ to $\mathcal{Y}$ is a tuple of algorithms$(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$, where:

- The randomized key-generation algorithm $\mathsf{Gen}()$ outputs a public-secret key pair (pk, sk).
- For a secret key sk and function input $\alpha \in \mathcal{X}$, the (possibly randomized) proving algorithm $\mathsf{Prove}_{sk}(\alpha) := \mathsf{Prove}(sk, \alpha)$ outputs a proof π.
- For a public key pk, function input $\alpha \in \mathcal{X}$, and proof π, the deterministic verification algorithm $\mathsf{Verify}_{pk}(\alpha, \pi) := \mathsf{Verify}(pk, \alpha, \pi)$ outputs some $\beta \in \mathcal{Y} \cup \{\perp\}$, where $\beta \in \mathcal{Y}$ denotes a valid proof with associated function output β, and $\perp \notin \mathcal{Y}$ is a distinguished value denoting an invalid proof.

The syntax presented above follows that of [GRPV22], and differs slightly from what is considered in some earlier works, where there is a separately defined evaluation procedure $\mathsf{Eval}_{sk}(\alpha) := \mathsf{Eval}(sk, \alpha)$ that takes a secret key sk and function input $\alpha \in \mathcal{X}$, and outputs a function value $\beta \in \mathcal{Y} \cup \{\perp\}$. The above-defined syntax directly yields such an evaluation algorithm, which runs $\pi \leftarrow \mathsf{Prove}_{sk}(\alpha)$ and outputs $\beta := \mathsf{Verify}_{pk}(\alpha, \pi)$. For ECVRF, evaluation can even be done *deterministically*, because the output β is the same regardless of the random choices made by Prove.

We require a VRF to have the following correctness property.

Definition 2.4 (Completeness). A VRF is *(perfectly) complete* if for a correctly generated key and proof, verification always succeeds. That is, for any input $\alpha \in \mathcal{X}$,

$$\Pr\left[\mathsf{Verify}_{pk}(\alpha, \pi) \in \mathcal{Y} : \begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}() \\ \pi \leftarrow \mathsf{Prove}_{sk}(\alpha) \end{array}\right] = 1.$$

We next consider various security properties for VRFs.

Definition 2.5 (Full Uniqueness). A VRF that uses one or more oracles is (Q, ϵ)-*fully unique* if any algorithm $\mathcal{A}$ with query complexity Q can produce a public key, a VRF input, and two valid proofs that yield different function outputs with probability at most ϵ. That is,

$$\mathbf{Adv}^{\text{f-uniq}}(\mathcal{A}) := \Pr\left[\begin{array}{c} \beta_0 := \mathsf{Verify}_{pk^*}(\alpha^*, \pi_0^*) \neq \perp \text{ and} \\ \beta_1 := \mathsf{Verify}_{pk^*}(\alpha^*, \pi_1^*) \neq \perp \text{ and} \\ \beta_0 \neq \beta_1 \end{array} : (pk^*, \alpha^*, \pi_0^*, \pi_1^*) \leftarrow \mathcal{A}()\right] \leq \epsilon.$$

Note that in the above definition, $\mathcal{A}$ takes no explicit input, but it has access to a random oracle (which is what the probability is taken over).

We note that the original uniqueness definition for a VRF is *perfect*, i.e., for any (possibly malformed) public key and any function input, there is at most a single function output for which a valid proof *exists*. However, the full uniqueness property is merely *computational*, i.e., it says that it is infeasible to find a violation of uniqueness.

Other previously defined properties of VRFs include (trusted or full) collision resistance, pseudorandomness, and unpredictability. We will not need any of these in this work, so we leave them undefined here and refer the interested reader to the prior works [MRV99, PWH+17].

2.4 ECVRF

Algorithm 1 formally defines the version of ECVRF of primary interest for this work. In brief, a secret key is a nonzero exponent $x \in \mathbb{Z}_q \setminus \{0\}$, and the corresponding public key is $X = B^x \in \mathbb{G}$. Each VRF input $\alpha \in \mathcal{X}$ maps to some $H \in \mathbb{G}$ via a hash function HTC, which stands for 'hash to curve' (see, e.g., [FHSS+22]). The prover computes $Z := H^x$ and proves that (B, X, H, Z), after cofactor clearing, is a Diffie–Hellman tuple. (This is done using a Fiat–Shamir-transformed variant of the Chaum–Pedersen protocol; see Section 4 for details.) The actual VRF output is a hash of Z after cofactor clearing.

Comparison to other versions. The ECVRF construction defined in Algorithm 1 very closely follows version 10 of [GRPV22], with the following main differences:[5]

- In [GRPV22], the blinding term r in Line 7 of Prove is not chosen uniformly at random, but instead is generated in a deterministic manner by applying a pseudorandom function to the secret key x and the hash digest H (hence the entire proof procedure is also deterministic). For simplicity, we treat r as uniformly random in our description and analysis.
- In [GRPV22], Verify has a 'key validation' option, which additionally checks that the public key X, after cofactor clearing, is not the identity element. While key validation is essential for certain properties of ECVRF (like collision resistance), it is not needed for any of the properties studied in this work, so for simplify we omit it from our presentation.

Other versions of [GRPV22], and its precursor [PWH+17], define the 'challenge' value c differently, by using different inputs to one or both of the hash functions HTC, H. Most notably, in response to our observations about the technical difficulties in proving uniqueness for versions 2 through 10 of [GRPV22], versions 11 and later define $c := \mathsf{H}(X, H, Z, R_B, R_H)$ on Line 8 (and they check this equality on Line 14).

[5] Another slight difference is that in [GRPV22], the input to HTC is more general: it consists of a 'salt' value together with α, where the salt is determined by the specific choice of ciphersuite (see [GRPV22, Section 7.9]). In every ECVRF ciphersuite defined in [GRPV22], the salt is simply the public key X, which matches our presentation.

Algorithm 1 Elliptic Curve VRF (ECVRF)

Public parameters:

- $\mathcal{X}, \mathcal{Y}$ respectively denote the domain and range of the VRF.
- $(\mathbb{G}, q, B)$ denotes a cyclic group of prime order q with generator B, which is a subgroup of a group $\mathbb{E}$ (for which checking membership is meant to be fast), and the cofactor $f = |\mathbb{E}|/|\mathbb{G}|$ is not divisible by q.
- $\mathsf{HTC}\colon \mathbb{E} \times \mathcal{X} \to \mathbb{G}$ and $\mathsf{H}\colon \mathbb{E}^4 \to \mathbf{H}$, where $\mathbf{H} \subseteq \mathbb{Z}_q$ is sufficiently large, are two hash functions (often modeled as random oracles).
- $\mathsf{E} : \mathbb{E} \to \mathcal{Y}$ is another hash function (not necessarily modeled as a random oracle).

Transformations between elements of $\mathbb{Z}_q$, $\mathbf{H}$, or $\mathbb{E}$ and their representations as bit strings are omitted, though we emphasize that canonical encodings and decodings are needed for non-malleability.

function $\mathsf{Gen}()$
  $x \leftarrow \mathbb{Z}_q \setminus \{0\}$; $X := B^x \in \mathbb{G}$
  return $(pk := X, sk := x)$
function $\mathsf{Prove}(x \in \mathbb{Z}_q, \alpha \in \mathcal{X})$
  $H := \mathsf{HTC}(X, \alpha) \in \mathbb{G}$
  $Z := H^x \in \mathbb{G}$
  $r \leftarrow \mathbb{Z}_q$; $R_B := B^r \in \mathbb{G}$; $R_H := H^r \in \mathbb{G}$
  $c := \mathsf{H}(H, Z, R_B, R_H) \in \mathbf{H}$
  $s := r + x \cdot c \in \mathbb{Z}_q$
  return $\pi := (Z, c, s)$
function $\mathsf{Verify}(X \in \mathbb{E}, \alpha \in \mathcal{X}, \pi = (Z \in \mathbb{E}, c \in \mathbf{H}, s \in \mathbb{Z}_q))$
  $H := \mathsf{HTC}(X, \alpha) \in \mathbb{G}$
  $R_B := B^s X^{-c} \in \mathbb{E}$; $R_H := H^s Z^{-c} \in \mathbb{E}$
  if $c = \mathsf{H}(H, Z, R_B, R_H)$ **then return** $\mathsf{E}(Z^f) \in \mathcal{Y}$ **else return** $\perp$

Note that here X is an explicit input to H, even though it is also used to derive the H-input $H := \mathsf{HTC}(X, \alpha)$. Our analysis in Sections 4 and 5 shows that properties like binding, non-malleability, and uniqueness can be proved even for the earlier versions of [GRPV22], though new ideas are needed. The differences in hashing do not substantially affect the prior proofs of other properties like pseudorandomness and collision resistance.

3 Relative Indifferentiability and Domain Extension

In this section, we put forth the notion of *indifferentiability relative to an auxiliary oracle*, or simply *relative indifferentiability*, in both the classical and quantum settings. This will be needed later in our analysis of the full uniqueness property of ECVRF (Section 4).

3.1 Indifferentiability Relative to an Auxiliary Oracle

Our definition of indifferentiability relative to an auxiliary oracle is a natural extension of the original definition from [MRH04]: all entities—the distinguisher $\mathcal{D}$, the simulator $\mathcal{S}$, and the construction C—additionally have access to some auxiliary oracle O. In the quantum setting (as considered in, e.g., [Zha19]), all oracles can be queried in superposition.

Definition 3.1. Let H' be a random function, and $\mathsf{C}^{\mathsf{O},\mathsf{H}}$ be a procedure with the same domain and range as H', which can query a (possibly stateful) oracle O and a random function H. We say that $\mathsf{C}^{\mathsf{O},\mathsf{H}}$ is (Q_D, Q_S, ϵ)-*indifferentiable from a random oracle relative to* O if there exists a simulator $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ with query complexity Q_S per invocation such that, for any distinguisher $\mathcal{D}$ with query complexity Q_D,

$$|\Pr[\mathcal{D}^{\mathsf{O},\mathsf{H},\mathsf{C}^{\mathsf{O},\mathsf{H}}} \text{ accepts}] - \Pr[\mathcal{D}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'},\mathsf{H}'} \text{ accepts}]| \leq \epsilon.$$

In ordinary indifferentiability, the simulator $\mathcal{S}$ gets to simulate (to the distinguisher $\mathcal{D}$) *all* the oracles to which the construction C has access. By contrast, in relative indifferentiability, *the simulator $\mathcal{S}$ does not simulate the auxiliary oracle* O; instead, $\mathcal{S}$ (and $\mathcal{D}$) can merely *query* O. This implies that relative indifferentiability is at least as strong as ordinary indifferentiability (for a corresponding query complexity), as long as O is computable. That is, if $\mathsf{C}^{\mathsf{O},\mathsf{H}}$ is (classically or quantumly) indifferentiable from a random oracle relative to O, then it is also (resp., classically or quantumly) indifferentiable from a random oracle in the ordinary sense. This is simply because, instead of $\mathcal{S}$ having O as an oracle, $\mathcal{S}$ can just implement O internally to answer O-queries for itself and the distinguisher.

Remark 3.2. Definition 3.1 is tailored to this work's focus on information-theoretic security, i.e., both the simulator $\mathcal{S}$ and distinguisher $\mathcal{D}$ can use unbounded computation; their number of queries is the only complexity measure. In the context of computational security, one may additionally require the simulator to be efficient, either asymptotically or concretely. All of the indifferentiability simulators considered in this work are efficient according to any reasonable notion, even when the distinguishers are not required to be.

Indistinguishability and consistency. Indifferentiablity is implied by the conjunction of two weaker notions called *indistinguishability* and *consistency*, as defined in [Zha19]. Here we adapt those definitions to work relative to an auxiliary oracle.

Definition 3.3. Let H, H', and O be as in Definition 3.1. A simulator $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ is (Q_D, ϵ)-*indistinguishable from a random oracle relative to* O if for any distinguisher $\mathcal{D}$ with query complexity Q_D,

$$|\Pr[\mathcal{D}^{\mathsf{O},\mathsf{H}} \text{ accepts}] - \Pr[\mathcal{D}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}} \text{ accepts}]| \leq \epsilon.$$

Definition 3.4. Let H' and O be as in Definition 3.1. A simulator $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ is (Q_D, ϵ)-*consistent for* C *relative to* O if, for any distinguisher $\mathcal{D}$ with query complexity Q_D,

$$|\Pr[\mathcal{D}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'},\mathsf{C}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}}} \text{ accepts}] - \Pr[\mathcal{D}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'},\mathsf{H}'} \text{ accepts}]| \leq \epsilon.$$

Lemma 3.5 (Adapted from [Zha19, Lemma 6]). *Let H' and $\mathsf{C}^{\mathsf{O},\mathsf{H}}$ be as in Definition 3.1, and suppose that C has query complexity $Q_C = (Q_{C,1}, Q_{C,2})$ per invocation. Then $\mathsf{C}^{\mathsf{O},\mathsf{H}}$ is $(Q_D, Q_S, \epsilon_1 + \epsilon_2)$-indifferentiable from a random oracle relative to O if there is a simulator $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ with query complexity Q_S per invocation that is both $((Q_{D,1} + Q_{C,1} \cdot Q_{D,3}, Q_{D,2} + Q_{C,2} \cdot Q_{D,3}), \epsilon_1)$-indistinguishable from a random oracle, and (Q_D, ϵ_2)-consistent for C, relative to O.*

The proof is an easy adaptation of the (straightforward) one given in [Zha19], so we only provide a brief sketch: the proof goes through one intermediate hybrid experiment where the distinguisher is given the oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{C}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}}$. It directly invokes consistency to show that this hybrid is indistinguishable from the 'ideal' experiment in Definition 3.1, and uses indistinguishability to show that the hybrid is also indistinguishable from the 'real' experiment. This latter connection uses a reduction that internally evaluates C using its

two oracles whenever the distinguisher queries its third oracle, which yields the query complexity from the indistinguishability hypothesis.

We stress that for the above lemma to apply, the *same* simulator $\mathcal{S}$ must be both indistinguishable and consistent. This is why these are defined as properties of the simulator, not the procedure C.

3.2 Find-Input Oracles

Definition 3.1 above introduces a more general notion of indifferentiability, which requires specifying an auxiliary oracle O. In this work, we focus on what we call *find-input* oracles. These implement a (classical) 'lazy' or (quantum) 'compressed' random oracle, and also have a second interface that exposes what is called the $\mathsf{FindInput}$ function. (As usual, in the quantum setting, this interface is accessible in superposition.) In essense, $\mathsf{FindInput}$ takes a value in the range of the oracle, and returns a previously queried input that maps to that range value, or a failure symbol if no such input exists. We first recall a few formalisms that will be used to define (both classical and quantum) find-input oracles.

Definition 3.6 (Database). Let $\mathcal{X}, \mathcal{Y}$ be two finite sets, and let $\perp \notin \mathcal{X} \cup \mathcal{Y}$ denote a distinguished value. A *database* $\mathbf{D}$ over domain $\mathcal{X}$ and range $\mathcal{Y}$ is an ordered list of pairs from $(\mathcal{X} \times \mathcal{Y}) \cup \{(\perp, \perp)\}$, where:[6]

- the pairs are sorted by their first entries (under some suitable ordering of $\mathcal{X}$),
- all $(\perp, \perp)$ pairs are at the end of the list,
- for each $x \in \mathcal{X}$, there is at most one $y \in \mathcal{Y}$ for which $(x, y) \in \mathbf{D}$; if there is such a y, we write $\mathbf{D}(x) = y$, otherwise we write $\mathbf{D}(x) = \perp$.

We say that $\mathbf{D}$ contains a *collision* if it has two pairs $(x, y), (x', y)$ for some *distinct* $x, x' \in \mathcal{X}$ and some $y \in \mathcal{Y}$.

Definition 3.7 (Database insertion). For a database $\mathbf{D}$ over domain $\mathcal{X}$ and range $\mathcal{Y}$ having at least one $(\perp, \perp)$ pair, and a pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ where $\mathbf{D}(x) = \perp$, define $\mathbf{D} \cup (x, y)$ to be the new database obtained by inserting (x, y) into $\mathbf{D}$ at the appropriate location (to maintain the sorted order), and removing one $(\perp, \perp)$.

We remark that the assumed existence and removal of a $(\perp, \perp)$ pair ensure that the database has the same size before and after the insertion operation, which is convenient in the quantum setting (though it is not needed in the classical setting).

Definition 3.8 ($\mathsf{FindInput}$). For a database $\mathbf{D}$ over domain $\mathcal{X}$ and range $\mathcal{Y}$, and some $y \in \mathcal{Y}$, the (classical) procedure $\mathsf{FindInput}(\mathbf{D}, y)$ outputs an element of $\mathcal{X}' := \mathcal{X} \cup \{\perp\}$ as follows: it checks whether there is an $x \in \mathcal{X}$ for which $(x, y) \in \mathbf{D}$. If so, it outputs the smallest such x; otherwise, it outputs $\perp$.[7]

Definition 3.9 (FILO). For finite domain $\mathcal{X}$ and range $\mathcal{Y}$, a classical *find-input lazy oracle* (FILO) is a stateful oracle $\mathsf{O} = (\mathsf{G}, \mathsf{FI}_{\mathsf{G}})$ that is initialized with an empty database $\mathbf{D}$ and provides two classical interfaces, G and FI_{G}, as follows:

- On query $\mathsf{G}(x)$ where $x \in \mathcal{X}$, first append a $(\perp, \perp)$ entry to $\mathbf{D}$. Then, if $\mathbf{D}(x) = \perp$, choose $y \leftarrow \mathcal{Y}$ and set $\mathbf{D} := \mathbf{D} \cup (x, y)$. Finally, return $\mathbf{D}(x)$.

[6] See Remark 3.11 below for a simpler alternative formulation that suffices for information-theoretic results.

[7] This definition of $\mathsf{FindInput}$ has some minor syntactic differences from the one given in [Zha19, Section 5.3], where the input is a pair (y, x_2), and the output is $(1, (x, x_2))$ when the search succeeds, and $(0, \mathbf{0})$ otherwise. Either version can trivially be constructed from the other, so they are equivalent. Our version is better suited to the definition of find-input oracles, because it does not involve any inputs to other oracles (namely, x_2).

- On query $\mathsf{FI}_{\mathsf{G}}(y)$ where $y \in \mathcal{Y}$, return $\mathsf{FindInput}(\mathbf{D}, y)$.

Queries to both interfaces are counted toward query complexity for this oracle.

In short, a FILO implements a lazy random oracle G, and also finds preimages of given G-outputs according to the query history thus far.

Definition 3.10 (FICO). For finite domain $\mathcal{X}$ and range $\mathcal{Y}$, a *find-input compressed oracle* (FICO) is a stateful oracle $\mathsf{O} = (\mathsf{G}, \mathsf{FI}_{\mathsf{G}})$ that is initialized with an empty database $\mathbf{D}$ and provides two interfaces, G and FI_{G}, defined as follows. However, the *classical* interface, which is needed by classical cryptographic constructions, is limited to G alone (following Section 2.1).

1. G is implemented as an ordinary 'compressed standard oracle' CStO (or equivalently, a 'compressed phase oracle' CPhsO) with (growing) database $\mathbf{D}$ in superposition, as defined in [Zha19, Section 3]. Essentially, CStO applies an efficient 'decompression' unitary called $\mathsf{StdDecomp}$ to the database, followed by the standard query unitary, followed by 'recompression' (which is actually identical to decompression, since it is an involution).

2. FI_{G} performs the unitary defined on the computational basis states as

$$|y, z\rangle \otimes |\mathbf{D}\rangle \mapsto |y, z \oplus \mathsf{FindInput}(\mathbf{D}, y)\rangle \otimes |\mathbf{D}\rangle$$

   for $y \in \mathcal{Y}$ and $z \in \mathcal{X}' = \mathcal{X} \cup \{\bot\}$, where $\mathcal{X}'$ is (without loss of generality) an abelian group with operation $\oplus$ and identity element $\bot$.

   Equivalently, FI_{G} can be defined to have a 'phase interface,' which performs the unitary defined by

$$|y, \chi\rangle \otimes |\mathbf{D}\rangle \mapsto \chi(\mathsf{FindInput}(\mathbf{D}, y)) \cdot |y, \chi\rangle \otimes |\mathbf{D}\rangle,$$

   where $\chi \in \widehat{\mathcal{X}'}$ is a character of $\mathcal{X}'$, i.e., a group homomorphism from $\mathcal{X}'$ to the complex unit circle.[8] Since χ outputs a scalar 'phase,' this interface can be seen as introducing the phase to either the query registers $|y, \chi\rangle$, or the database $\mathbf{D}$ itself.

As with a FILO, queries to both interfaces are counted toward the query complexity for this oracle.

In short, a FICO implements a compressed oracle, and also gives superposition access to preimages according to the query history. Zhandry [Zha19] shows that having (quantum) access to a compressed oracle alone—equivalently, having access to a FICO without using its find-input interface—is identical to having quantum access to a random oracle.

Remark 3.11. We note that because all of our results relating to compressed oracles are information theoretic (i.e., they depend only on the adversary's query complexity and not its running time), we could alternatively use the computationally inefficient but technically simpler approach of representing find-input oracles using the full 'value tables' of *partial* functions, as explicated in [Unr21, Section 3.1]. In this approach, the state of a FILO (or FICO) reflects a partial function (in superposition) from $\mathcal{X}$ to $\mathcal{Y}$, which is represented by an $|\mathcal{X}|$-dimensional vector over $\mathcal{Y} \cup \{\bot\}$ (initialized to the empty function), and $\mathsf{FindInput}$ is defined in the obvious way. We adhere to the efficient compact representations from [Zha19] in order to make all of our (quantum) algorithms efficient, which may be useful in future work.

[8] Recall that the character group $\widehat{\mathcal{X}'}$ is isomorphic to $\mathcal{X}'$, but non-canonically. The equivalence of FI_{G}'s standard and phase interfaces follows by applying the (inverse) quantum Fourier transform before and after each query.

Quantum query bounds. Several known bounds on quantum query complexity for random oracles, which were re-proved using compressed oracles in [Zha19], also extend easily to FICOs. Here we state this formally for collision finding, which will be used in our quantum indifferentiability proofs. Similar results hold for the optimality of Grover search [Gro96] and for more general relation-finding, including the k-sum problem, by adapting the arguments in [Zha19, Section 4] to FICOs.

Lemma 3.12 (Adapted from [Zha19, Theorem 2]). *Let $\mathsf{O} = (\mathsf{G}, \mathsf{FI}_{\mathsf{G}})$ be a FICO with range $\mathcal{Y}$ and (growing) database $\mathbf{D}$. After an adversary makes Q queries to G and any number of queries to FI_{G}, we have*

$$\|P \cdot |\psi\rangle\| \leq O(\sqrt{Q^3/|\mathcal{Y}|}),$$

where $|\psi\rangle$ is the joint state of the adversary and the oracle, and P is the projection onto the span of basis states where $\mathbf{D}$ contains a collision. In particular, the database will contain a collision with probability $O(Q^3/|\mathcal{Y}|)$.

Proof summary. The proof closely parallels the ones from [Zha19, Theorems 1 and 2], so we do not repeat the details. The only difference is that we need to address the effect of FI_{G}-queries. It suffices to observe that an FI_{G}-query does not increase $\|P \cdot |\psi\rangle\|$, because it does not alter the database $\mathbf{D}$; it only changes the phases in the joint state $|\psi\rangle$. Formally, we have that $\|P \cdot \mathsf{FI}_{\mathsf{G}} \cdot |\psi\rangle\| = \|P \cdot |\psi\rangle\|$. Finally, the prior analysis of the effect of G-queries goes through unchanged here, because it is agnostic to phases. □

3.3 Relative Indifferentiability of a Domain Extender

Let $\mathsf{O} = (\mathsf{H}_1, \mathsf{FI}_{\mathsf{H}_1})$ be a (classical or quantum) find-input oracle with domain $\mathcal{X}_1$ and range $\mathcal{Y}_1$, and $\mathsf{H}_2 \colon \mathcal{Y}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$ be a random oracle. Define $\mathsf{C}^{\mathsf{O},\mathsf{H}_2} \colon \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$ as

$$\mathsf{C}^{\mathsf{O},\mathsf{H}_2}(x_1, x_2) = \mathsf{H}_2(\mathsf{H}_1(x_1), x_2).$$

Notice that C does not query the $\mathsf{FI}_{\mathsf{H}_1}$ interface of O, so it can be instantiated with just ordinary (classical or quantumly accessible) random oracles $\mathsf{H}_1, \mathsf{H}_2$. Clearly, C's query complexity per invocation is $Q_C = (1, 1)$ in the classical setting, and $Q_C = (2, 1)$ in the quantum setting; here the first entry is 2 because C also needs to 'uncompute' the intermediate value $\mathsf{H}_1(x_1)$ after invoking H_2.

The ordinary indifferentiability of $\mathsf{C}^{\mathsf{H}_1,\mathsf{H}_2}$ from a random oracle is proved in [CDMP05, Lemma 1] for the classical setting, and in [Zha19, Theorem 4] for the quantum setting. Here we extend these results to show that C satisfies our stronger notion of indifferentiability relative to the find-input oracle O, where only H_2 is simulated.

Our proofs of relative indifferentiability are mainly 'refactorings' of the proofs of ordinary indifferentiability from [CDMP05, Zha19]. The key observation is that, while ordinary indifferentiability allows the simulator to simulate H_1 in whatever fashion it chooses, the cited works' simulators actually use very little of this power: they merely implement H_1 as an ordinary (lazy or compressed) oracle, and suitably 'record' the distinguisher's queries to it. This is in contrast to their simulations of H_2, which use more sophisticated strategies that rely on having suitable access to the H_1 database. Our main insight is that both H_1 and this database access can be encapsulated as a find-input oracle and made 'external' to the simulator (instead of being simulated by it), and the proofs can be adapted to this setting of relative indifferentiability. We note that this adaptation is not entirely trivial, because the *distinguisher* also gets find-input access to H_1, so we need to extend the proof techniques to show that this extra power does not help the distinguisher.

We also point out that in both the classical and quantum settings, the simulators from our proofs of indifferentiability never query the H_1 interface of oracle O. Furthermore, looking ahead, in our analysis of

the full uniqueness of ECVRF (Section 4), the distinguisher never queries the $\mathsf{FI}_{\mathsf{H}_1}$ interface of O. Therefore, for our application, it would suffice to define the indifferentiability experiments so that $\mathcal{S}$ has access to $\mathsf{FI}_{\mathsf{H}_1}$ but not H_1, and $\mathcal{D}$ has access to H_1 but not $\mathsf{FI}_{\mathsf{H}_1}$. We choose to give both $\mathcal{S}$ and $\mathcal{D}$ full access to O in because this yields a more natural and general extension of indifferentiability, which may be useful in other contexts.

3.3.1 Classical Indifferentiability

We start with the classical setting, proving the following theorem.

Theorem 3.13. *When $\mathsf{O} = (\mathsf{H}_1, \mathsf{FI}_{\mathsf{H}_1})$ is a FILO and H_2 is a classical random oracle, the domain extender C is $(Q_D = (Q_{D,1}, Q_{D,2}, Q_{D,3}), Q_S = (1,1), \epsilon)$-indifferentiable from a random oracle relative to O (Definition 3.1), where*

$$\epsilon = \frac{2(Q_{D,1} + Q_{D,2})Q_{D,3} + (Q_{D,1} + Q_{D,3})^2}{2|\mathcal{Y}_1|} \leq \frac{3Q_D^2}{4|\mathcal{Y}_1|}.$$

Proof. We need to construct a simulator $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ that has access to O and a random oracle $\mathsf{H}' \colon \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$, and simulates answers to a distinguisher $\mathcal{D}$'s H_2-queries. For simplicity, assume that $\mathcal{D}$ never repeats a query to H_1, nor to its second or third oracles; this is without loss of generality because $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}'$ are functions, and $\mathcal{S}$ can also be implemented as a function using memorization. Note that $\mathcal{D}$ may repeat queries to $\mathsf{FI}_{\mathsf{H}_1}$, because it is stateful and its answers may change as queries are made to H_1.

The simulator $\mathcal{S}$ is defined as follows: on query (y, x_2), it queries $x_1 := \mathsf{FI}_{\mathsf{H}_1}(y)$. If $x_1 \neq \perp$ (i.e., if $x_1 \in \mathcal{X}_1$), then $\mathcal{S}$ queries $\mathsf{H}'(x_1, x_2)$ and forwards the response to $\mathcal{D}$. Otherwise, $\mathcal{S}$ returns a uniformly random element in $\mathcal{Y}_2$. Clearly, $\mathcal{S}$ has query complexity $Q_S = (1,1)$ per invocation.

The indistinguishability and consistency of $\mathcal{S}$ (with suitable bounds, and using the fact that $\mathsf{C}^{\mathsf{H}_1,\mathsf{H}_2}$ has classical query complexity $Q_C = (1,1)$ per invocation) are shown below in Lemmas 3.14 and 3.15, respectively. By Lemma 3.5, this establishes the claim. □

Lemma 3.14. *For any $\tilde{Q} = (\tilde{Q}_1, \tilde{Q}_2)$, the simulator $\mathcal{S}$ defined in the proof of Theorem 3.13 is $(\tilde{Q}, 0)$-indistinguishable (Definition 3.3), i.e., the simulation is perfect.*

Proof. Following the definition of indistinguishability, consider a distinguisher $\widetilde{\mathcal{D}}$ with query complexity $\tilde{Q}$ that is given access either to oracles O, H_2 (the 'real' experiment) or to oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}$ (the 'ideal' experiment). As above, we assume without loss of generality that $\widetilde{\mathcal{D}}$ never repeats a query to H_1 or to its second oracle (whether H_2 or $\mathcal{S}$).

We next observe that the database $\mathbf{D}_1$ for O grows identically in both experiments. This is because $\mathsf{FI}_{\mathsf{H}_1}$- and H_2-queries do not change $\mathbf{D}_1$, and $\mathcal{S}$ never queries H_1, hence it also does not change $\mathbf{D}_1$. Therefore, $\mathbf{D}_1$ can be changed only by an explicit H_1-query made by $\widetilde{\mathcal{D}}$. This implies that answers to $\widetilde{\mathcal{D}}$'s O-queries are answered identically in both experiments, since the answers are determined solely by $\mathbf{D}_1$.

Now we consider $\widetilde{\mathcal{D}}$'s queries to its second oracle, i.e., H_2 in the real experiment and $\mathcal{S}$ in the ideal one. Consider a query (y, x_2), which by assumption has never been queried before. In the real experiment, the answer is a fresh uniformly random element of $\mathcal{Y}_2$. In the ideal experiment there are two cases:

- If there is no x_1 such that $\mathbf{D}_1(x_1) = y$, then $\mathcal{S}$ returns a fresh uniformly random element $y \leftarrow \mathcal{Y}_2$ as the answer.

- Otherwise, there was a previous $\mathsf{H}_1(x_1)$ query (by $\widetilde{\mathcal{D}}$) whose output is y, i.e., $\mathbf{D}_1(x_1) = y$. Recall that $\mathcal{S}$ queries $\mathsf{H}'(x_1, x_2)$ and forwards the answer to $\widetilde{\mathcal{D}}$. We claim that $\mathcal{S}$ has not queried $\mathsf{H}'(x_1, x_2)$ previously, so the answer is a fresh uniformly random element of $\mathcal{Y}_2$. To see this, observe that any previous $\mathsf{H}'(x_1, x_2)$ query by $\mathcal{S}$ must have been induced by a previous (y', x_2) query to $\mathcal{S}$ (by $\widetilde{\mathcal{D}}$) where $y' = \mathbf{D}(x_1) = y$. Therefore, $\widetilde{\mathcal{D}}$ previously queried (y, x_2), which contradicts the hypothesis that it is a new query.

So, in both experiments, each of $\widetilde{\mathcal{D}}$'s queries to its second oracle is answered by a fresh uniform element of $\mathcal{Y}_2$. This completes the proof. □

Lemma 3.15. *The simulator $\mathcal{S}$ defined in the proof of Theorem 3.13 is (Q_D, ϵ)-consistent (Definition 3.4), where Q_D, ϵ are as in the statement of Theorem 3.13.*

Proof. Following the definition of consistency, consider a distinguisher $\widetilde{\mathcal{D}}$ with query complexity $Q_D = (Q_{D,1}, Q_{D,2}, Q_{D,3})$ that is given access either to oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{C}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}}$ (the 'real' experiment) or to oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{H}'$ (the 'ideal' experiment), where $\mathsf{H}' \colon \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$ is a random oracle.

The rest of the proof proceeds via a series of hybrid experiments, starting from the ideal experiment and ending with the real experiment.

Hybrid 0 is the ideal experiment.

Hybrid 1 is the ideal experiment, except that whenever $\widetilde{\mathcal{D}}$ queries its third oracle on (x_1, x_2), the experiment first queries $y := \mathsf{H}_1(x_1)$ before answering with $\mathsf{H}'(x_1, x_2)$. Note that this adds (x_1, y) to the database $\mathbf{D}_1$ underlying O, if it was not already present.

Claim 3.16. The difference between the probabilities that $\widetilde{\mathcal{D}}$ accepts in Hybrids 0 and 1 is at most

$$\frac{(Q_{D,1} + Q_{D,2}) \cdot Q_{D,3}}{|\mathcal{Y}_1|} \leq \frac{Q_D^2}{4|\mathcal{Y}_1|}.$$

Let Query be the event that there ever exists an $x_1 \in \mathcal{X}_1$ such that $\widetilde{\mathcal{D}}$ has queried its third oracle on (x_1, x_2) for some x_2, and $\mathsf{FI}_{\mathsf{H}_1}(y)$ for $y = \mathbf{D}_1(x_1)$ has been queried (either directly, or indirectly by querying $\mathcal{S}(y, x_2')$ for some arbitrary x_2'), but $\widetilde{\mathcal{D}}$ has not queried $\mathsf{H}_1(x_1)$ directly. We claim that if Query does not occur, then $\widetilde{\mathcal{D}}$'s views in Hybrids 0 and 1 are identically distributed, so the distinguishing advantage between Hybrids 0 and 1 is at most $\Pr[\mathsf{Query}]$, and that this is at most the bound stated in the claim.

To see this, suppose that $\widetilde{\mathcal{D}}$ has queried its third oracle on some (x_1, x_2) and has not (yet) queried $\mathsf{H}_1(x_1)$ directly. Then $(x_1, y = \mathbf{D}_1(x_1))$ is in $\mathbf{D}_1$ in Hybrid 1, but not in Hybrid 0. However, the only way this difference can possibly affect $\mathcal{D}$'s view is if $\mathsf{FI}_{\mathsf{H}_1}(y)$ is eventually queried (either directly or indirectly), because no other queries' answers are affected by this additional database entry. In addition, since y is uniformly random and independent of $\widetilde{\mathcal{D}}$'s view until $\mathsf{FI}_{\mathsf{H}_1}(y)$ is queried, every (direct or indirect) query to $\mathsf{FI}_{\mathsf{H}_1}$ has probability at most $1/|\mathcal{Y}_1|$ of making Query occur due to this particular x_1. Since $\mathsf{FI}_{\mathsf{H}_1}$ is queried at most $Q_{D,1} + Q_{D,2}$ times, and there are at most $Q_{D,3}$ values of x_1 that may cause Query to happen, the claimed bound on $\Pr[\mathsf{Query}]$ follows by the union bound.

Hybrid 2 is the same as Hybrid 1, except that whenever $\widetilde{\mathcal{D}}$ queries its third oracle on (x_1, x_2), causing the experiment to query $y := \mathsf{H}_1(x_1)$, the experiment sets $x_1' := \mathsf{FI}_{\mathsf{H}_1}(y)$ and returns $\mathsf{H}'(x_1', x_2)$. (Observe that $x_1' \neq \perp$ because $(x_1, y) \in \mathbf{D}_1$ when $\mathsf{FI}_{\mathsf{H}_1}(y)$ is called.)

Claim 3.17. The difference between the probabilities that $\widetilde{\mathcal{D}}$ accepts in Hybrids 1 and 2 is at most

$$\frac{(Q_{D,1}+Q_{D,3})^2}{2|\mathcal{Y}_1|} \leq \frac{Q_D^2}{2|\mathcal{Y}_1|}.$$

Let Collision be the event that there ever exist distinct $x_1, x_1' \in \mathcal{X}_1$ such that $\mathbf{D}_1(x_1) = \mathbf{D}_1(x_1')$. Clearly, $\widetilde{\mathcal{D}}$'s views in Hybrids 1 and 2 are identically distributed unless Collision happens, because in the absence of any collisions, $x_1' = x_1$ in the above procedure. Since an entry can be added to $\mathbf{D}_1$ only when $\widetilde{\mathcal{D}}$ queries H_1 or its third oracle, the distinguishing advantage between Hybrids 1 and 2 is at most $\Pr[\mathsf{Collision}]$, which is bounded by the expression in the above claim.

Finally, we argue that Hybrid 2 is identical to the real experiment. In the latter, $\widetilde{\mathcal{D}}$'s third oracle is $\mathsf{C}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}}$, where recall that $\mathsf{C}(x_1,x_2) = \mathcal{S}(\mathsf{H}_1(x_1),x_2)$. Unrolling this according to the definition of $\mathcal{S}$, this first queries $y := \mathsf{H}_1(x_1)$, then queries $x_1' := \mathsf{Fl}_{\mathsf{H}_1}(y)$, and since $x_1' \neq \perp$ because (x_1,y) is in the database, the answer is $\mathsf{H}'(x_1',x_2)$. This is exactly Hybrid 2, as claimed. This completes the proof of Lemma 3.15. □

Comparison with [CDMP05]. We briefly discuss the differences between our proof of Theorem 3.13 and that of [CDMP05, Lemma 1], which states that C satisfies ordinary indifferentiability.

In the security experiment for the ordinary indifferentiability of $\mathsf{C}^{\mathsf{H}_1,\mathsf{H}_2}$, the distinguisher $\mathcal{D}$ has access to H_1 but not $\mathsf{Fl}_{\mathsf{H}_1}$, and $\mathcal{S}$ additionally gets to simulate H_1. In the proof from [CDMP05], $\mathcal{S}$ internally implements H_1 via lazy sampling, maintaining a database $\mathbf{D}_1$ so that it can compute $\mathsf{Fl}_{\mathsf{H}_1}$ on its own whenever $\mathcal{D}$ makes an H_2-query. By contrast, our simulator does not simulate H_1, and to handle H_2 queries it explicitly queries $\mathsf{Fl}_{\mathsf{H}_1}$.

Another important difference is the following: observe that in both [CDMP05] and here, $\mathcal{D}$ can conditionally induce an H_1-query by querying its third oracle (either C or H'), which in the real experiment can change the database $\mathbf{D}_1$ but in the ideal experiment cannot. (Note that $\mathcal{D}$ does not receive the output from H_1.) Since $\mathcal{D}$ has access to $\mathsf{Fl}_{\mathsf{H}_1}$ in our setting, we must take into account the possibility that $\mathcal{D}$ tries to distinguish the two experiments by querying $\mathsf{Fl}_{\mathsf{H}_1}$ on the output of H_1 for such an induced query. This is handled by Claim 3.16, and is the reason why our concrete security bound is slightly worse (by a small constant factor) than the one in [CDMP05].

3.3.2 Quantum Indifferentiability

We now turn to the quantum setting, and prove the following theorem.

Theorem 3.18. *When $\mathsf{O} = (\mathsf{H}_1, \mathsf{Fl}_{\mathsf{H}_1})$ is a FICO and H_2 is a quantumly accessible random oracle, $\mathsf{C}^{\mathsf{O},\mathsf{H}_2}$ is $(Q_D = (Q_{D,1}, Q_{D,2}, Q_{D,3}), Q_S = (2,1), \epsilon)$-indifferentiable from a random oracle relative to O, where*

$$\epsilon = O(Q_D^2/\sqrt{|\mathcal{Y}_1|}).$$

A more refined bound on ϵ can be obtained from Lemmas 3.19 and 3.20 below. The constant factors hidden by the $O(\cdot)$ notation are explicit and moderate, and can be extracted from the proofs given in [Zha19].

Proof. We define a simulator $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ that simulates quantum access to an oracle $\mathsf{H}_2 \colon \mathcal{Y}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$ as follows. It internally implements a random function $\tilde{H} \colon \mathcal{Y}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$ as a compressed oracle, and answers H_2-queries by applying the unitary defined by the following action on basis states:

$$|(y_1,x_2),z\rangle \mapsto \begin{cases} |(y_1,x_2), z \oplus \mathsf{H}'(x_1,x_2)\rangle \text{ if } x_1 := \mathsf{Fl}_{\mathsf{H}_1}(y_1) \neq \perp \\ |(y_1,x_2), z \oplus \tilde{\mathsf{H}}(y_1,x_2)\rangle \text{ otherwise.} \end{cases}$$

This unitary is straightforward to implement with a single query to each of $\mathsf{H}', \tilde{\mathsf{H}}$ and two queries to $\mathsf{FI}_{\mathsf{H}_1}$, almost exactly as done in [Zha19, Appendix B.4]. The only difference is that the previous simulator's local FindInput computation (and uncomputation) are here implemented by querying $\mathsf{FI}_{\mathsf{H}_1}$. So, $\mathcal{S}$'s query complexity per invocation is $Q_S = (2, 1)$.

The indistinguishability and consistency of $\mathcal{S}$ (with suitable bounds, and using the fact that $\mathsf{C}^{\mathsf{H}_1,\mathsf{H}_2}$ has quantum query complexity $Q_C = (2, 1)$ per invocation) are shown below in Lemmas 3.19 and 3.20, respectively. By Lemma 3.5, this establishes the claim. □

The following two lemmas show the relative indistinguishability and consistency of $\mathcal{S}$. They closely parallel [Zha19, Lemmas 8 and 13], which show the ordinary versions of these properties for an analogous simulator. The proofs even use the same hybrid experiments as in [Zha19], except that here the distinguisher additionally has (quantum) access to $\mathsf{FI}_{\mathsf{H}_1}$. It is straightforward to extend the analysis from the prior proofs to handle this setting; we give the modified proofs in Appendix A.

Lemma 3.19. *For any $\tilde{Q} = (\tilde{Q}_{D,1}, \tilde{Q}_{D,2})$, the simulator $\mathcal{S}$ from the proof of Theorem 3.18 is $(\tilde{Q}, \epsilon)$-indistinguishable (Definition 3.3), where*

$$\epsilon = O(\tilde{Q}_{D,1} \cdot (\tilde{Q}_{D,1}^{1/2} + \tilde{Q}_{D,2})/\sqrt{|\mathcal{Y}_1|}).$$

Lemma 3.20. *The simulator $\mathcal{S}$ from the proof of Theorem 3.18 is (Q_D, ϵ)-consistent (Definition 3.4), where Q_D is as in the statement and*

$$\epsilon = O((Q_{D,1} + Q_{D,3})^{3/2}/\sqrt{|\mathcal{Y}_1|}) = O(Q_D^{3/2}/\sqrt{|\mathcal{Y}_1|}).$$

4 Full Uniqueness of ECVRF

In this section we show that ECVRF unconditionally has full uniqueness (Definition 2.5) against both classical *and quantum* attackers, in the random-oracle model (ROM) and the quantumly accessible random-oracle model (QROM), respectively. To achieve this, we proceed along several steps.

In Section 4.1 we recall the necessary background on proof systems. Then in Section 4.2 we give (a slight variant of) the Chaum–Pedersen Σ-protocol for proving equality of discrete logarithms [CP92], along with a self-contained proof of its soundness in our setting. In Section 4.3 we apply the Fiat–Shamir transformation to obtain a non-interactive proof system, and (unconditionally) obtain the soundness of its verifier in the ROM and QROM using the approach of [FS86, BR93] and a theorem of [DFMS19], respectively. However, the resulting non-interactive proof does not quite match the one implicit in ECVRF, due to differences in how the hashing is done, and prior indifferentiability theorems are not sufficient (for the reasons given in the introduction). To bridge this gap, in Section 4.4 we invoke the theorems on relative indifferentiability from Section 3.3. Finally, in Section 4.5 we show that the soundness of the non-interactive proof implies full uniqueness of ECVRF.

4.1 Proof Systems

As background, here we recall the notion of a Σ-*protocol* and a *noninteractive proof system*, and the definitions of *soundness* for them.

Definition 4.1 (Σ-protocol). A Σ-*protocol* for a language $\mathcal{L}$, with *challenge space* $\mathbf{H}$, is a three-message interactive proof system consisting of a prover $P = (P_0, P_1)$ and a deterministic verifier V.[9] For a given statement x and a witness w, the protocol proceeds as follows:

1. P computes a commitment $R \leftarrow P_0(x, w)$.
2. A uniformly random challenge $c \leftarrow \mathbf{H}$ is chosen and given to P.
3. P then generates a response $s \leftarrow P_1(c)$.
4. $V(x, R, c, s)$ either accepts or rejects.

A variety of security properties are often associated with Σ-protocols and other proof systems, such as (honest-verifier) zero knowledge, special and simulation soundness, etc. In this work, we only need the notion of ordinary soundness (we do not even explicitly need completeness). For a comprehensive description of other properties, including in the quantum setting, see [Unr17].

Definition 4.2 (Soundness, Σ-protocol). A Σ-protocol (or just its verifier) for a language $\mathcal{L}$ has *soundness error* ϵ if no (computationally unbounded) algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, acting as a prover, can cause the verifier to accept an invalid statement (of $\mathcal{A}$'s choice) with probability more than ϵ. That is,

$$\mathbf{Adv}_V^{\text{sound}}(\mathcal{A}) := \Pr\left[\begin{array}{c} V(x, R, c, s) \text{ accepts} \\ \text{and } x \notin \mathcal{L} \end{array} : \begin{array}{c} (x, R) \leftarrow \mathcal{A}_0() \\ c \leftarrow \mathbf{H} \\ s \leftarrow \mathcal{A}_1(c) \end{array}\right] \leq \epsilon.$$

Note that the above definition is statistical, i.e., it places no restrictions on the adversary's running time.

Definition 4.3 (Non-interactive proof system). A *non-interactive proof system* for a language $\mathcal{L}$ is a pair of algorithms (P, V), where:

- Given a statement x and a witness w, the prover $P(x, w)$ outputs a proof π.
- Given a statement x and a proof π, the deterministic verifier $V(x, \pi)$ either accepts or rejects.

Definition 4.4 (Soundness, non-interactive proof system). A non-interactive proof system (P, V) (or just its verifier V) for a language $\mathcal{L}$ that uses one or more oracles is (Q, ϵ)-*sound* if no (computationally unbounded) algorithm $\mathcal{A}$ with query complexity Q can cause the verifier to accept an invalid statement (of $\mathcal{A}$'s choice) with probability more than ϵ. That is,

$$\mathbf{Adv}_V^{\text{sound}}(\mathcal{A}) := \Pr\left[\begin{array}{c} V(x, \pi) \text{ accepts} \\ \text{and } x \notin \mathcal{L} \end{array} : (x, \pi) \leftarrow \mathcal{A}() \right] \leq \epsilon.$$

Note that in the above definition, $\mathcal{A}$ takes no explicit input, but it has access to one or more (typically random) oracles, which is what the probability is taken over.

Fiat–Shamir. In Algorithm 2 we recall the Fiat–Shamir transformation [FS86], which transforms a Σ-protocol into a non-interactive proof system.

The following result addresses the soundness of the Fiat–Shamir transformation on Σ-protocols, in the (Q)ROM. The first part, which concerns the ROM, is from [FS86, BR93]. The second part, which concerns the QROM, is from [DFM20, Theorem 3] (improving on [DFMS19, Theorem 8]). We point out that all these

[9] The component P_1 represents a 'continuation' of P_0, and implicitly has access to all of its inputs and random choices.

Algorithm 2 Fiat–Shamir transformation of Σ-protocol (P, V) with challenge space $\mathbf{H}$

Public parameters: Random oracle H whose range is $\mathbf{H}$.

function $P_{\mathrm{FS}}(x, w)$
  $R \leftarrow P_0(x, w)$; $c := \mathsf{H}(x, R)$; $s \leftarrow P_1(c)$
  return $\pi = (R, s)$
function $V_{\mathrm{FS}}(x, \pi)$
  Parse $\pi = (R, s)$ (and reject if this fails)
  $c := \mathsf{H}(x, R)$
  return $V(x, R, c, s)$

results 'relativize,' i.e., they hold even in the presence of other, possibly stateful or quantumly accessible, oracle(s): the reductions simply pass along all queries to, and answers from, these extra oracles without using them in any other way.[10]

Proposition 4.5. Suppose that a Σ-protocol has soundness error ϵ in the presence of some (possibly stateful or quantumly accessible) oracle(s). Then for any $Q \geq 0$, and in the presence of the same oracle(s), the protocol's Fiat–Shamir transformation (Algorithm 2) is:

1. $(Q, (Q+1)\epsilon)$-sound when H is a classical random oracle.
2. $(Q, (2Q+1)^2\epsilon = O(Q^2\epsilon))$-sound when H is a quantumly accessible random oracle.

4.2 Chaum–Pedersen Protocol

As in ECVRF (Algorithm 1), fix a cyclic group $(\mathbb{G}, q, B)$ of known prime order q with known generator B, where $\mathbb{G}$ is a subgroup of a group $\mathbb{E}$ (for which checking membership is meant to be fast) having cofactor $f = |\mathbb{E}|/|\mathbb{G}|$ that is not divisible by q. Throughout this section, for a group element $G \in \mathbb{E}$, let $\hat{G} := G^f \in \mathbb{G}$. In addition, fix a challenge space $\mathbf{H} \subseteq \mathbb{Z}_q$, which should be sufficiently large for soundness.

The Chaum–Pedersen Σ-protocol (slightly generalized to our setting of two groups $\mathbb{G} \subseteq \mathbb{E}$) is for statements of the form $(X \in \mathbb{E}, H \in \mathbb{G}, Z \in \mathbb{E})$, and it proves membership in the language $\mathcal{L} = \mathcal{L}_{\mathcal{R}}$ of the relation

$$\mathcal{R} := \{((X, H, Z), x) : \hat{X} = B^x \text{ and } \hat{Z} = H^x\}.$$

In other words, (X, H, Z) is in the language exactly when $(B, \hat{X}, H, \hat{Z}) \in \mathbb{G}^4$ is a Diffie-Hellman tuple.

The protocol proceeds as follows. The prover and verifier are given a statement $(X \in \mathbb{E}, H \in \mathbb{G}, Z \in \mathbb{E})$, and the prover is additionally given its witness x (when the statement is in the language).

1. The prover chooses $r \leftarrow \mathbb{Z}_q$ and lets its commitment be $R_B := B^r \in \mathbb{G}$, $R_H := H^r \in \mathbb{G}$.
2. A uniformly random challenge $c \leftarrow \mathbf{H}$ is chosen and given to the prover.
3. The prover lets its response be $s := r + x \cdot c \in \mathbb{Z}_q$.

[10] We remark that Unruh [Unr17, Corollary 36] proved a similar result for the QROM. However, Unruh's reduction does not attack the soundness of the underlying Σ-protocol, but instead solves a kind of search problem on the QRO, in a manner that for technical reasons is not suitable for our setting. In brief, we need a reduction that 'relativizes' in the presence of an auxiliary *stateful* oracle, without making any additional queries to it (only the ones made by the adversary itself). This is the case for the reduction from [DFMS19], but not for the one from [Unr17] in our context. Furthermore, the concrete security bound in [Unr17] is slightly worse than that in [DFMS19].

4. The verifier, given the statement and R_B, R_H, c, s, accepts if $R_B = B^s X^{-c}$ and $R_H = H^s Z^{-c}$; otherwise, it rejects.[11]

We remark that the protocol is complete (i.e., the prover causes the verifier to accept) for the *sublanguage* $\mathcal{L} \cap \mathbb{G}^3$, i.e., tuples $(X, H, Z) \in \mathcal{L}$ where all three components are elements of $\mathbb{G} \subseteq \mathbb{E}$. This is sufficient for the completeness of ECVRF, because all honestly generated elements are in $\mathbb{G}$. However, completeness of ECVRF can also be seen on its own, irrespective of this Σ-protocol.

Lemma 4.6. *The (variant) Chaum–Pedersen protocol has soundness error* $1/|\mathbf{H}|$ *(Definition 4.2).*

Proof. Consider an arbitrary statement $(X \in \mathbb{E}, H \in \mathbb{G}, Z \in \mathbb{E}) \notin \mathcal{L}$ that is not in the language. We separately consider two cases: either H is the identity element $e \in \mathbb{G}$, or it is not. In both cases we show that, once a potentially malicious prover sends its initial message, there exists at most one challenge value $c \in \mathbf{H}$ for which the proof can be completed to make the verifier accept. Because the challenge is chosen uniformly and independently of the prover's initial message, the soundness error is $1/|\mathbf{H}|$.

If $H = e$, then $(X, H, Z) \notin \mathcal{L}$ implies that $\hat{Z} \neq e$, so $\hat{Z}$ is a generator of $\mathbb{G}$ (because $\mathbb{G}$ is a prime-order cyclic group). Suppose than an adversarial prover sends some $R_H \in \mathbb{E}$ in its initial message. (Note that $R_H \in \mathbb{E}$ without loss of generality, because otherwise the verification equations cannot be satisfied.) A uniformly random challenge $c \in \mathbf{H}$ is chosen, and the prover generates some s. In order for the verifier to accept, by raising the second verification equation to the f power and rearranging, it must be the case that

$$\hat{R}_H \hat{Z}^c = \hat{H}^s = e,$$

and thus $c = -\log_{\hat{Z}} \hat{R}_H$. (Note that s has no effect here.) So, the prover cannot make the verifier accept unless this unique value of c is chosen as the challenge.

Next, if $H \neq e$, then H is a generator of $\mathbb{G}$, and $\log_B \hat{X} \neq \log_H \hat{Z}$. Since f is coprime with q, there exist distinct $x, x' \in \mathbb{Z}_q$ such that $\hat{X} = \hat{B}^x$ and $\hat{Z} = \hat{H}^{x'}$. Suppose that an adversarial prover sends some $R_B, R_H \in \mathbb{E}$ as its initial message, and let $r = \log_{\hat{B}} \hat{R}_B$, $r' = \log_{\hat{H}} \hat{R}_H$. (Note that r, r' need not be equal, but they are uniquely defined because $\hat{B}, \hat{H}$ are generators of $\mathbb{G}$.) A uniformly random challenge $c \in \mathbf{H}$ is then chosen. In order for there to *exist* some $s \in \mathbb{Z}_q$ that would make the verifier accept, by raising the verification equations to the f power and rearranging, it must be the case that

$$\hat{B}^s = \hat{R}_B \hat{X}^c = \hat{B}^r (\hat{B}^x)^c = \hat{B}^{r + x \cdot c}$$
$$\hat{H}^s = \hat{R}_H \hat{Z}^c = \hat{H}^{r'} (\hat{H}^{x'})^c = \hat{H}^{r' + x' \cdot c}$$

and thus $s = r + x \cdot c = r' + x' \cdot c \pmod{q}$. This implies that $c = (r - r')(x' - x)^{-1} \in \mathbb{Z}_q$, which is well defined because $x \neq x'$ and q is prime. So, the prover cannot make the verifier accept unless this unique value of c is chosen as the challenge. □

In order to link the above protocol to the ECVRF construction, from now on we consider statements of the form $(X \in \mathbb{E}, \alpha \in \mathcal{X}, Z \in \mathbb{E})$, which define the associated statements $(X, H = \mathsf{HTC}(X, \alpha), Z)$, where $\mathsf{HTC}\colon \mathbb{E} \times \mathcal{X} \to \mathbb{G}$ is the oracle used in ECVRF. In other words, we consider the 'language'

$$\mathcal{L}_{\mathsf{HTC}} := \{(X, \alpha, Z) : (X, H = \mathsf{HTC}(X, \alpha), Z) \in \mathcal{L}\}.$$

Note that, since HTC in our context is usually treated as a FILO or FICO, which are defined 'lazily,' membership in $\mathcal{L}_{\mathsf{HTC}}$ may not be determined until $H = \mathsf{HTC}(X, \alpha)$ is queried (classically). Therefore, the

[11] Note that because $B, X, H, Z \in \mathbb{E}$, these checks implicitly guarantee that $R_B, R_H \in \mathbb{E}$ as well.

soundness experiments from Definitions 4.2 and 4.4 implicitly perform this query at the *very end*, when testing membership. But also note that a typical verifier, including all the ones considered in this work, would have already performed this query when deciding whether to accept.

The Chaum–Pedersen protocol can be naturally extended to a Σ-protocol for $\mathcal{L}_{\mathsf{HTC}}$, simply by augmenting the prover and verifier to compute $H := \mathsf{HTC}(X, \alpha)$ and then proceed as before. The following lemma shows that this protocol is sound *even if the attacker also gets find-input access to* HTC (see Section 3.2); we need to give the attacker this extra power when we use our results on relative indifferentiability in Section 4.4 below.

Lemma 4.7. *For any query complexity Q_{O}, the described Σ-protocol for $\mathcal{L}_{\mathsf{HTC}}$ is*

1. *$(Q_{\mathsf{O}}, 1/|\mathbf{H}|)$-sound (Definition 4.4) in the presence of a FILO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$.*
2. *$(Q_{\mathsf{O}}, 1/|\mathbf{H}| + O(Q_{\mathsf{O}}/\sqrt{|\mathbb{G}|}))$-sound in the presence of a FICO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$.*

We note that in the additive $O(Q_{\mathsf{O}}/\sqrt{|\mathbb{G}|})$ term from Item 2 above, Q_{O} can be replaced by just the number of $\mathsf{FI}_{\mathsf{HTC}}$-queries made between when the adversarial prover outputs its chosen statement (X, α, Z) and when it outputs the final message of its attempted proof. In addition, the term may not be tight, and could potentially be improved or even eliminated.

Proof. Let $\mathcal{A}'$ be an attacker against the soundness of the above extended protocol for $\mathcal{L}_{\mathsf{HTC}}$, which has query complexity Q_{O} to the FILO/FICO. We transform it into an attacker $\mathcal{A}$ against the ordinary Chaum–Pedersen protocol for $\mathcal{L}$.

We start with the classical case. The attacker $\mathcal{A}$ internally implements a FILO and uses it to answer the queries of $\mathcal{A}'$. When $\mathcal{A}'$ outputs a chosen statement (X, α, Z) that it will attempt to prove, $\mathcal{A}$ internally queries $H = \mathsf{HTC}(X, \alpha)$, with the modification that if this query was not previously made by $\mathcal{A}'$, then this query-response pair is *not* added to the FILO database until $\mathcal{A}'$ queries (X, α), if ever.[12] Then $\mathcal{A}$ outputs (X, H, Z) as the statement it will attempt to prove. When $\mathcal{A}'$ outputs the initial message of its proof, $\mathcal{A}$ outputs the same; then $\mathcal{A}$ receives a challenge $c \in \mathbf{H}$, which it forwards to $\mathcal{A}'$; then $\mathcal{A}'$ outputs the final message of its attempted proof, and $\mathcal{A}$ outputs the same. By inspection, it is straightforward to see that $\mathcal{A}$ perfectly simulates the soundness attack game to $\mathcal{A}'$, and that $\mathcal{A}$ succeeds in its attack game exactly when $\mathcal{A}'$ succeeds in its own, which establishes the claim.

The quantum case is more subtle: because $\mathcal{A}'$ can make $\mathsf{FI}_{\mathsf{HTC}}$- and HTC-queries (in superposition) after sending its chosen statement (X, α, Z), it is not obvious whether the (classical) query $\mathsf{HTC}(X, \alpha)$ can be moved from the end of the experiment to the point where $\mathcal{A}'$ outputs the statement. Fortunately, we show that this is indeed the case.

Hybrid 0 is the original soundness experiment. Recall that a classical FICO query $H := \mathsf{HTC}(X, \alpha)$ is explicitly made at the end of the experiment, after $\mathcal{A}'$ has finished.

Hybrid 1 is the same as Hybrid 0, except that the aforementioned query is made earlier, right after $\mathcal{A}'$ outputs its chosen statement (X, α, Z).

Claim 4.8. The difference between the probabilities that $\mathcal{A}'$ succeeds in Hybrids 0 and 1 is $O(Q_{\mathsf{O}}/\sqrt{|\mathbb{G}|})$.

First observe that the timing of the *measurements* associated with the classical FICO query in question has no effect on the success probabilities, because the input (X, α) is already classical, and the measured

[12] This behavior ensures that the find-input queries of $\mathcal{A}'$ are answered exactly as in the real experiment, which does not explicitly query $\mathsf{HTC}(X, \alpha)$ until after $\mathcal{A}'$ finishes.

output is in a private register of the experiment. In addition, any of the adversary's measurements between outputting its statement and its final message can be deferred to the latter point in time, again without affecting its success probability. So, it suffices to analyze the effect of moving the underlying (quantum) HTC-query.

In Hybrid 0, between when $\mathcal{A}'$ outputs (X, α, Z) and the experiment's $\mathsf{HTC}(X, \alpha)$ query, $\mathcal{A}'$ may make some HTC- and $\mathsf{FI}_{\mathsf{HTC}}$-queries. Note that any two HTC-queries on different registers commute with each other (this can be seen because the $\mathsf{StdDecomp}$ operations at the end of the first query and beginning of the second query cancel out, and the internal query steps on the decompressed database trivially commute), and any HTC-query and $\mathsf{FI}_{\mathsf{HTC}}$-query $O(1/\sqrt{|\mathbb{G}|})$-'almost commute', using [Zha19, Lemma 7]. (This is because the only steps in an HTC-query that change the FICO database are the two $\mathsf{StdDecomp}$ operations, which $O(1/\sqrt{|\mathbb{G}|})$-almost commute with $\mathsf{FI}_{\mathsf{HTC}}$.) Since $\mathcal{A}'$ makes at most Q_{O} $\mathsf{FI}_{\mathsf{HTC}}$-queries, the claim follows.

Claim 4.9. The probability that $\mathcal{A}'$ succeeds in Hybrid 1 is at most $1/|\mathbf{H}|$.

This can be seen similarly to the classical case. The reduction $\mathcal{A}$ works as defined there, except that when $\mathcal{A}'$ outputs (X, α, Z), it simply queries $\mathsf{HTC}(X, \alpha)$ without any modifications. Combining the two claims above yields the lemma. □

4.3 Fiat–Shamir-Transformed Proof

Algorithm 3 Verifier (and optimization) from the Fiat–Shamir-transformed proof system for $\mathcal{L}_{\mathsf{HTC}}$

Public parameters: hash functions $\mathsf{HTC}\colon \mathbb{E} \times \mathcal{X} \to \mathbb{G}$ and $\mathsf{H}'\colon \mathbb{E} \times \mathcal{X} \times \mathbb{E}^3 \to \mathbf{H}$, where $\mathbf{H} \subseteq \mathbb{Z}_q$.

function $V_{\mathrm{FS}}((X \in \mathbb{E}, \alpha \in \mathcal{X}, Z \in \mathbb{E}), \pi = (R_B \in \mathbb{E}, R_H \in \mathbb{E}, s \in \mathbb{Z}_q))$
  $H := \mathsf{HTC}(X, \alpha) \in \mathbb{G}$
  $c := \mathsf{H}'(X, \alpha, Z, R_B, R_H) \in \mathbf{H}$
  if $R_B = B^s X^{-c}$ and $R_H = H^s Z^{-c}$ **then** accept **else** reject
function $V'_{\mathrm{FS}}((X \in \mathbb{E}, \alpha \in \mathcal{X}, Z \in \mathbb{E}), \pi' = (c \in \mathbf{H}, s \in \mathbb{Z}_q))$
  $H := \mathsf{HTC}(X, \alpha) \in \mathbb{G}$
  $R_B := B^s X^{-c} \in \mathbb{E}$, $R_H := H^s Z^{-c} \in \mathbb{E}$
  if $c = \mathsf{H}'(X, \alpha, Z, R_B, R_H)$ **then** accept **else** reject

We now make the Σ-protocol for $\mathcal{L}_{\mathsf{HTC}}$ non-interactive via the Fiat–Shamir transform. Algorithm 3 gives the verifier from the transformed proof system, along with an optimized version where the proof contains the challenge c instead of the commitment R_B, R_H. (Because we are concerned only with *soundness* here, from this point on we deal only with verifiers, and omit any treatment of provers.) Combining Lemma 4.7 with Proposition 4.5—which, to recall, holds even relative to stateful oracles like FILOs and FICOs—we get the following results on the soundness of the Fiat–Shamir-transformed verifiers.

Lemma 4.10. *For any $Q = (Q_{\mathsf{O}}, Q_{\mathsf{H}'})$, the verifier V_{FS} from Algorithm 3 is:*

1. *$(Q, \epsilon = (Q_{\mathsf{H}'} + 1)/|\mathbf{H}|)$-sound for a FILO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a random oracle H';*
2. *$(Q, \epsilon = O(Q_{\mathsf{H}'}^2/|\mathbf{H}| + Q_{\mathsf{H}'}^2 Q_{\mathsf{O}}/\sqrt{|\mathbb{G}|}))$-sound for a FICO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a quantumly accessible random oracle H'.*

Lemma 4.11. *V'_{FS} is $((Q_{\mathsf{O}}, Q_{\mathsf{H}'}), \epsilon)$-sound for a FILO or FICO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and random oracle H', if V_{FS} is $((Q_{\mathsf{O}} + 1, Q_{\mathsf{H}'}), \epsilon)$-sound for the same oracles.*

Proof. For any adversary $\mathcal{A}'$ with query complexity $(Q_{\mathsf{O}}, Q_{\mathsf{H}'})$ against the soundness of V'_{FS}, we construct an adversary $\mathcal{A}$ with query complexity $(Q_{\mathsf{O}}+1, Q_{\mathsf{H}'})$ that attacks the soundness of V_{FS}: $\mathcal{A}$ runs $\mathcal{A}'$, forwarding the queries of $\mathcal{A}'$ to $\mathcal{A}$'s own oracles (and forwarding the answers back to $\mathcal{A}'$). When $\mathcal{A}'$ outputs a statement-proof pair $((X \in \mathbb{E}, \alpha \in \mathcal{X}, Z \in \mathbb{E}), \pi' = (c \in \mathbf{H}, s \in \mathbb{Z}_q))$, $\mathcal{A}$ queries $H := \mathsf{HTC}(X, \alpha) \in \mathbf{H}$, computes $R_B := B^s X^{-c} \in \mathbb{E}$ and $R_H := H^s Z^{-c} \in \mathbb{E}$, and outputs the statement-proof pair $((X, \alpha, Z), \pi = (R_B, R_H, s))$. By inspection, $\mathcal{A}$ makes at most one more HTC-query than $\mathcal{A}'$ does, perfectly simulates the attack game against V'_{FS} to $\mathcal{A}'$, and succeeds in its attack game against V_{FS} whenever $\mathcal{A}'$ succeeds in its own, which establishes the claim. □

4.4 Using Relative Indifferentiability

Algorithm 4 Non-interactive proof verifier used in ECVRF

Public parameters: hash functions $\mathsf{HTC}\colon \mathbb{E} \times \mathcal{X} \to \mathbb{G}$ and $\mathsf{H}\colon \mathbb{E}^4 \to \mathbf{H}$, where $\mathbf{H} \subseteq \mathbb{Z}_q$.

1: **function** $V_{\mathrm{ECVRF}}((X \in \mathbb{E}, \alpha \in \mathcal{X}, Z \in \mathbb{E}), \pi = (c \in \mathbf{H}, s \in \mathbb{Z}_q))$
2: $\quad H := \mathsf{HTC}(X, \alpha) \in \mathbb{G}$
3: $\quad R_B := B^s X^{-c} \in \mathbb{E}$, $R_H := H^s Z^{-c} \in \mathbb{E}$
4: $\quad$ **if** $c = \mathsf{H}(H, Z, R_B, R_H)$ **then** accept **else** reject

We now show the soundness of the non-interactive proof verifier V_{ECVRF} implicit in ECVRF, which is given in Algorithm 4. The only difference between V'_{FS} and V_{ECVRF} is that the former uses a separate independent hash function H' to derive the challenge c, whereas the latter uses a composition of H and HTC. This difference is addressed using our results on relative indifferentiability from Section 3, which we use to show that any attack against the soundness of V_{ECVRF} implies a similarly effective attack against the soundness of V'_{FS}.

We stress that, in contrast to the usual applications of indifferentiability, the verifiers V_{ECVRF} and V'_{FS} use the 'inner' function HTC for purposes beyond just its composition with H, namely, the value R_H is derived from $H = \mathsf{HTC}(X, \alpha)$ and is used to verify the proof. Because of this, we cannot allow an indifferentiability simulator to simulate HTC, because this is not allowed in the soundness attack game against V'_{FS} (indeed, allowing it might even make V'_{FS} unsound). Our notion of *relative* indifferentiability circumvents this difficulty, by making HTC external to the simulator, and allowing it to simulate only H using its access to HTC and H'. This lets us construct a legal attack against V'_{FS} from any attack against V_{ECVRF}, as shown in the following lemma.

Lemma 4.12. *V_{ECVRF} (Algorithm 4) is $(Q = (Q_{\mathsf{O}}, Q_{\mathsf{H}}), \epsilon)$-sound for a FILO (respectively, FICO) $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a random oracle (resp., quantumly accessible random oracle) H, if V'_{FS} is (Q', ϵ')-sound for the same O and a random oracle (resp., quantumly accessible random oracle) H', where*

$$Q' = (Q'_{\mathsf{O}}, Q'_{\mathsf{H}'}) = (Q_{\mathsf{O}} + Q_{\mathsf{H}}, Q_{\mathsf{H}}) \,,\ \epsilon = \epsilon' + \frac{3(Q'+2)^2}{4|\mathbb{G}|} = \epsilon' + O(Q^2/|\mathbb{G}|)$$

in the classical setting, and

$$Q' = (Q'_{\mathsf{O}}, Q'_{\mathsf{H}'}) = (Q_{\mathsf{O}} + 2Q_{\mathsf{H}}, Q_{\mathsf{H}}) \,,\ \epsilon = \epsilon' + O(Q^2/\sqrt{|\mathbb{G}|})$$

in the quantum setting.

Proof. The only difference between V'_{FS} and V_{ECVRF} is that c is checked against $\mathsf{H}'(X, \alpha, Z, R_B, R_H)$ in V'_{FS}, and against $\mathsf{C}^{\mathsf{HTC},\mathsf{H}}(X, \alpha, Z, R_B, R_H) := \mathsf{H}(\mathsf{HTC}(X, \alpha), Z, R_B, R_H)$ in V_{ECVRF}. We handle this distinction using the results on relative indifferentiability from Section 3.3.

We give a detailed proof for the quantum setting; the classical setting follows essentially identically from Theorem 3.13 with the appropriate bounds. By Theorem 3.18, there exists an algorithm $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$ with query complexity $Q_S = (2, 1)$ per invocation such that, for any distinguisher $\mathcal{D}$ with query complexity $Q_D = (Q_{D,1}, Q_{D,2}, Q_{D,3})$,

$$|\Pr[\mathcal{D}^{\mathsf{O},\mathsf{H},\mathsf{C}^{\mathsf{HTC},\mathsf{H}}} \text{ accepts}] - \Pr[\mathcal{D}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'},\mathsf{H}'} \text{ accepts}]| = O(Q_D^2/\sqrt{|\mathbb{G}|}), \tag{4.1}$$

where $\mathsf{H} \colon \mathbb{E}^4 \to \mathbf{H}$ is a random oracle. Recall that $\mathcal{S}$ simulates only oracle H.

Let $\mathcal{A}^{\mathsf{O},\mathsf{H}}$ be a quantum adversary with query complexity $Q = (Q_{\mathsf{O}}, Q_{\mathsf{H}})$ that attacks the soundness of V_{ECVRF}. We construct a quantum reduction $\mathcal{R}^{\mathsf{O},\mathsf{H}'}$ that attacks the soundness of V'_{FS}. Essentially, $\mathcal{R}$ is just the composition of $\mathcal{A}$ and $\mathcal{S}$. More specifically, $\mathcal{R}$ internally runs both $\mathcal{A}$ and $\mathcal{S}$, forwarding any of $\mathcal{A}$'s O-queries to its own O oracle, and answering any of $\mathcal{A}$'s H-queries using $\mathcal{S}$ (by forwarding $\mathcal{S}$'s queries to its own O or H' oracle, as appropriate). Finally, $\mathcal{R}$ outputs whatever statement-proof pair $\mathcal{A}$ outputs. Observe that $\mathcal{R}$'s query complexity is $(Q'_{\mathsf{O}}, Q'_{\mathsf{H}'}) = (Q_{\mathsf{O}} + 2Q_{\mathsf{H}}, Q_{\mathsf{H}})$, because it merely forwards all of $\mathcal{A}$'s O-queries to its own O, and each of $\mathcal{A}$'s H-queries is handled by $\mathcal{S}$ making two $\mathsf{FI}_{\mathsf{HTC}}$-queries and one H'-query.

In order to relate the advantages of $\mathcal{A}$ and $\mathcal{R}$ in their respective attack games, we define and analyze a quantum distinguisher $\mathcal{D}$ that aims to distinguish between the tuples of oracles $(\mathsf{O}, \mathsf{H}, \mathsf{C}^{\mathsf{HTC},\mathsf{H}})$ and $(\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{H}')$, as follows: $\mathcal{D}$ internally runs $\mathcal{A}$, forwarding $\mathcal{A}$'s O- and H-queries to its own first and second oracles (respectively), until $\mathcal{A}$ outputs a statement-proof pair $(x = (X, \alpha, Z), \pi = (c, s))$. Then, $\mathcal{D}$ checks whether $x \in \mathcal{L}_{\mathsf{HTC}}$, i.e., whether $(X, H = \mathsf{HTC}(X, \alpha), Z) \in \mathcal{L}$, and rejects if so.[13] Next, $\mathcal{D}$ checks the 'validity' of the proof: it computes $R_B = B^s X^{-c}$, $R_H = H^s Z^{-c}$, checks whether its third oracle outputs c when queried on (X, α, Z, R_B, R_H), and accepts if this holds, rejecting otherwise. Clearly, $\mathcal{D}$'s query complexity is $Q_D = (Q_{D,1}, Q_{D,2}, Q_{D,3}) = (Q_{\mathsf{O}} + 1, Q_{\mathsf{H}}, 1)$: it makes at most Q_{O} and Q_{H} queries to its first and second oracles (respectively) while running $\mathcal{A}$, one more query to its first oracle in order to check membership in $\mathcal{L}_{\mathsf{HTC}}$, and one query to its third oracle to check the proof's validity.

We now analyze $\mathcal{D}$ in the two experiments and relate it to the advantages of $\mathcal{A}$ and $\mathcal{R}$. Observe by inspection that if $\mathcal{D}$'s oracles are $\mathsf{O}, \mathsf{H}, \mathsf{C}^{\mathsf{HTC},\mathsf{H}}$ (the 'real' experiment), then $\mathcal{D}$ perfectly simulates the soundness attack game against V_{ECVRF} to $\mathcal{A}$, and accepts exactly when $\mathcal{A}$ succeeds (this is where we use the fact that $\mathcal{D}$ rejects when $x \in \mathcal{L}_{\mathsf{HTC}}$). So,

$$\Pr[\mathcal{D}^{\mathsf{O},\mathsf{H},\mathsf{C}^{\mathsf{HTC},\mathsf{H}}} \text{ accepts}] = \mathbf{Adv}^{\text{sound}}_{V_{\mathrm{ECVRF}}}(\mathcal{A}).$$

On the other hand, if $\mathcal{D}$'s oracles are $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{H}'$, then observe by inspection that $\mathcal{D}$, together with $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$, simulates to $\mathcal{A}$ exactly the same experiment as $\mathcal{R}^{\mathsf{O},\mathsf{H}'}$ does. Moreover, the checks that $\mathcal{D}$ performs on the statement-proof pair output by $\mathcal{A}$ (equivalently, by $\mathcal{R}$) are exactly those that define the success condition in the attack game against the soundness of V'_{FS} (in part, because $\mathcal{D}$'s third oracle is H'). Therefore,

$$\Pr[\mathcal{D}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'},\mathsf{H}'} \text{ accepts}] = \mathbf{Adv}^{\text{sound}}_{V'_{\mathrm{FS}}}(\mathcal{R}).$$

By Equation (4.1) and the hypothesis on V'_{FS}, we therefore have $\mathbf{Adv}^{\text{sound}}_{V_{\mathrm{ECVRF}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sound}}(\mathcal{R}) + O(Q^2/\sqrt{|\mathbb{G}|}) \leq \epsilon' + O(Q^2/\sqrt{|\mathbb{G}|})$, as needed. □

[13] Note that $\mathcal{D}$ can perform this check by brute force, because it can be computationally unbounded; its query complexity is the only relevant metric here.

4.5 Full Uniqueness

Lemma 4.13. *For any $Q = (Q_{\mathsf{O}}, Q_{\mathsf{H}})$, ECVRF (Algorithm 1) is (Q, ϵ)-fully unique (Definition 2.5) for a FILO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a random oracle H (respectively, for a FICO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a random oracle H), if V_{ECVRF} is $(Q, \epsilon/2)$-sound.*

Proof. Let $\mathcal{A}^{\mathsf{O},\mathsf{H}}$ be an adversary with query complexity Q that attacks the full uniqueness of ECVRF. We construct a reduction $\mathcal{R}^{\mathsf{O},\mathsf{H}}$ that attacks the soundness of V_{ECVRF} as follows: $\mathcal{R}$ runs $\mathcal{A}$, and forwards $\mathcal{A}$'s O- and H-queries to $\mathcal{R}$'s own oracles, relaying the answers back to $\mathcal{A}$. On $\mathcal{A}$'s output $(X, \alpha, \pi_0 = (Z_0, c_0, s_0), \pi_1 = (Z_1, c_1, s_1))$, $\mathcal{R}$ chooses $b \leftarrow \{0,1\}$ uniformly at random, sets $(Z, c, s) := (Z_b, c_b, s_b)$, and outputs $((X, \alpha, Z), \pi = (c, s))$.[14] Clearly, $\mathcal{R}$ perfectly simulates the full-uniqueness attack game to $\mathcal{A}$, and its query complexity is the same as $\mathcal{A}$'s.

We now relate the advantages of $\mathcal{R}$ and $\mathcal{A}$ in their respective attack games. Suppose that $\mathcal{A}$ succeeds in the full-uniqueness attack game against ECVRF. Then we claim that at least one of the two statements $(X, \alpha, Z_b) \notin \mathcal{L}_{\mathsf{HTC}}$, so in this event $\mathcal{R}$ has probability at least $1/2$ of succeeding in its game. Indeed:

- For either choice of $b \in \{0,1\}$ we have $\mathsf{Verify}_X(\alpha, (Z, c, s)) \neq \perp$, i.e., $c = \mathsf{H}(H, Z, R_B, R_H)$, where $H = \mathsf{HTC}(X, \alpha)$, $R_B = B^s X^{-c}$, and $R_H := H^s Z^{-c}$. So, $V_{\text{ECVRF}}((X, \alpha, Z), (c, s))$ accepts.
- Also, $\beta_0 = \mathsf{E}(\hat{Z}_0) \neq \mathsf{E}(\hat{Z}_1) = \beta_1$, so $\hat{Z}_0 \neq \hat{Z}_1$, and thus (X, α, Z_0) and (X, α, Z_1) cannot be both in $\mathcal{L}_{\mathsf{HTC}}$. This is because for any fixed $X \in \mathbb{E}, \alpha \in \mathcal{X}$, we have $\hat{X} = B^x$ for some unique $x \in \mathbb{Z}_q$, so $\hat{Z} = \mathsf{HTC}(X, \alpha)^x \in \mathbb{G}$ is uniquely defined when $(X, \alpha, Z) \in \mathcal{L}_{\mathsf{HTC}}$.

Combining the above, we have that $\mathbf{Adv}^{\text{sound}}_{V_{\text{ECVRF}}}(\mathcal{R}) \geq \frac{1}{2}\mathbf{Adv}^{\text{f-uniq}}(\mathcal{A})$, and the lemma follows. □

Finally, our ultimate theorem on the full uniqueness of ECVRF follows by Lemmas 4.10 to 4.13 and parameter bookkeeping. Note that the full uniqueness in the ordinary (quantumly accessible) random-oracle model—i.e., *without* any find-input access—is an immediate corollary of this theorem.

Theorem 4.14. *For any $Q = (Q_{\mathsf{O}}, Q_{\mathsf{H}})$, ECVRF (Algorithm 1) is (Q, ϵ)-fully unique (Definition 2.5) for a FILO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a random oracle H, where*

$$\epsilon = \frac{2(Q_{\mathsf{H}} + 1)}{|\mathbf{H}|} + \frac{3(Q_{\mathsf{O}} + 2Q_{\mathsf{H}} + 2)^2}{2|\mathbb{G}|} = 2(Q_{\mathsf{H}} + 1)/|\mathbf{H}| + O(Q^2/|\mathbb{G}|),$$

and for a FICO $\mathsf{O} = (\mathsf{HTC}, \mathsf{FI}_{\mathsf{HTC}})$ and a quantumly accessible random oracle H, where

$$\epsilon = O(Q^2/|\mathbf{H}| + Q^3/\sqrt{|\mathbb{G}|}).$$

Proof. In the classical setting,

- By Lemmas 4.10 and 4.11, V'_{FS} is $(Q' = (Q'_{\mathsf{O}}, Q'_{\mathsf{H}'}), (Q'_{\mathsf{H}'} + 1)/|\mathbf{H}|)$-sound for any Q'.
- Then by Lemma 4.12 (setting $Q'_{\mathsf{O}} = Q_{\mathsf{O}} + Q_{\mathsf{H}}$ and $Q'_{\mathsf{H}'} = Q_{\mathsf{H}}$), V_{ECVRF} is

$$\Big((Q_{\mathsf{O}}, Q_{\mathsf{H}}), \frac{Q_{\mathsf{H}} + 1}{|\mathbf{H}|} + \frac{3(Q_{\mathsf{O}} + 2Q_{\mathsf{H}} + 2)^2}{4|\mathbb{G}|}\Big)\text{-sound.}$$

[14] Alternatively, $\mathcal{R}$ could determine which $(X, \alpha, Z_b) \notin \mathcal{L}_{\mathsf{HTC}}$ by brute force. This would improve its advantage by a factor of two, at the cost of one more HTC-query and an enormous amount of computation.

- Finally, by Lemma 4.13, ECVRF is

$$\left((Q_{\mathsf{O}}, Q_{\mathsf{H}}), \frac{2(Q_{\mathsf{H}}+1)}{|\mathbf{H}|} + \frac{3(Q_{\mathsf{O}}+2Q_{\mathsf{H}}+2)^2}{2|\mathbb{G}|}\right)\text{-fully unique.}$$

In the quantum setting,

- By Lemmas 4.10 and 4.11, V'_{FS} is $(Q' = (Q'_{\mathsf{O}}, Q'_{\mathsf{H}'}), O((Q'_{\mathsf{H}'})^2/|\mathbf{H}| + (Q'_{\mathsf{H}'})^2 Q'_{\mathsf{O}}/\sqrt{|\mathbb{G}|}))$-sound for any Q'.
- Then by Lemma 4.12 (setting $Q'_{\mathsf{O}} = Q_{\mathsf{O}} + 2Q_{\mathsf{H}}$ and $Q'_{\mathsf{H}'} = Q_{\mathsf{H}}$), V_{ECVRF} is

$$\left((Q_{\mathsf{O}}, Q_{\mathsf{H}}), O(Q^2/|\mathbf{H}| + Q^3/\sqrt{|\mathbb{G}|})\right)\text{-sound.}$$

- Finally, by Lemma 4.13, ECVRF is

$$\left((Q_{\mathsf{O}}, Q_{\mathsf{H}}), O(Q^2/|\mathbf{H}| + Q^3/\sqrt{|\mathbb{G}|})\right)\text{-fully unique.}$$ □

5 Binding and Non-Malleability of ECVRF

In this section we consider the new notions of *(trusted or full) binding* and *(strong) non-malleability* for VRFs. In Section 5.1 we formally define these concepts, and relate them to each other (Theorem 5.4). Then in Section 5.2 we show that assuming the collision resistance of its hash functions, ECVRF satisfies full binding, even against quantum attacks. Finally, in Section 5.3 we show that against classical (but not quantum) attacks and assuming the intractability of the discrete logarithm problem, ECVRF additionally satisfies (strong) non-malleability.

5.1 New Security Notions

Here we introduce the notions of binding and non-malleability for VRFs.

Binding. Binding says, informally, that a proof uniquely determines (computationally) the input (or input and public key) for which it is valid, if any. This is closely related to notions of binding that have recently been defined for signature schemes [BCJZ21, CGN20, CDF+21]. We mainly consider two notions at opposite ends of a spectrum: the weaker one, called *trusted binding*, requires that the public key is generated correctly; the stronger one, called *full binding*, allows the adversary to generate public keys on its own, possibly maliciously.

Definition 5.1 (Trusted binding). A VRF is (t, Q, ϵ)-*trusted binding* if no algorithm $\mathcal{A}$ running in time at most t and with query complexity Q, given oracle access to the proving procedure (possibly among other oracles), can produce two different function inputs and one proof that is valid for both inputs with probability more than ϵ. That is,

$$\mathbf{Adv}^{\text{t-bind}}(\mathcal{A}) := \Pr\left[\begin{array}{c}\mathsf{Verify}_{pk}(\alpha_0^*, \pi^*) \neq \perp \text{ and} \\ \mathsf{Verify}_{pk}(\alpha_1^*, \pi^*) \neq \perp \text{ and} \\ \alpha_0^* \neq \alpha_1^*\end{array} : \begin{array}{c}(pk, sk) \leftarrow \mathsf{Gen}() \\ (\alpha_0^*, \alpha_1^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{Prove}_{sk}(\cdot)}(pk)\end{array}\right] \leq \epsilon. \tag{5.1}$$

We could also consider a stronger notion that directly gives the adversary $\mathcal{A}$ the secret key sk, instead of oracle access to Prove_{sk}. However, we will not use this notion in this work.

Definition 5.2 (Full binding). A VRF is (t, ϵ)-*full binding* if no algorithm $\mathcal{A}$ running in time at most t can produce two public key-input pairs and one proof that is valid for both pairs with probability more than ϵ. That is,

$$\mathbf{Adv}^{\text{f-bind}}(\mathcal{A}) := \Pr\left[\begin{array}{c} \mathsf{Verify}(pk_0^*, \alpha_0^*, \pi^*) \neq \perp \text{ and} \\ \mathsf{Verify}(pk_1^*, \alpha_1^*, \pi^*) \neq \perp \text{ and} \\ (pk_0^*, \alpha_0^*) \neq (pk_1^*, \alpha_1^*) \end{array} : (pk_0^*, \alpha_0^*, pk_1^*, \alpha_1^*, \pi^*) \leftarrow \mathcal{A}()\right] \leq \epsilon. \tag{5.2}$$

Note that in the above definition, $\mathcal{A}$ takes no explicit input. This is identical to the situation with collision resistance as described in Section 2.2, and we treat it in the same way.

We can also consider the weaker notion of *full input binding*, which is the same as full binding but additionally requires that $pk_0^* = pk_1^*$, and hence $\alpha_0^* \neq \alpha_1^*$ (so, a proof uniquely determines the function input, but not necessarily the public key). However, we will not use this notion anywhere in this work.

(Strong) non-malleability. Non-malleability says that without knowing the secret key, it is infeasible to produce a valid proof (for an input of one's choice) that is different from all the proofs provided by the legitimate prover. We note that this alone does *not* rule out the possibility of a legitimate proof being valid for a *different* input than the one for which it was produced; we address that issue below with the notion of *strong* non-malleability. We also emphasize that non-malleability does not prevent a legitimate prover, who knows the secret key, from producing different proofs for the same input (indeed, this is easy to do in the ECVRF construction).

Definition 5.3 (Non-malleability). A VRF is (t, Q, ϵ)-*non-malleable* if no algorithm $\mathcal{A}$ running in time at most t and with query complexity Q, given oracle access to the proving procedure (possibly among other oracles), can produce a valid input-proof pair where the proof was not output by the oracle with probability more than ϵ. That is,

$$\mathbf{Adv}^{\text{NM}}(\mathcal{A}) := \Pr\left[\begin{array}{c} \mathsf{Verify}_{pk}(\alpha^*, \pi^*) \neq \perp \text{ and} \\ \pi^* \text{ was not a response to any query} \end{array} : \begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}() \\ (\alpha^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{Prove}_{sk}(\cdot)}(pk) \end{array}\right] \leq \epsilon. \tag{5.3}$$

Strong non-malleability is defined in the same way via $\mathbf{Adv}^{\text{SNM}}(\mathcal{A})$, where the event of interest is that (α^*, π^*) was not a query-response pair of $\mathcal{A}$'s oracle.

Obtaining strong non-malleability. Observe that an adversary that breaks strong non-malleability must do so in one of two ways: either by giving a 'new' valid proof that it did not receive from the legitimate prover, thus breaking non-malleability, or by giving an 'old' proof (that it received from the prover) that is valid for a different input than the one(s) that yielded that proof, thus breaking trusted binding. The following theorem shows that this intuition can be formalized: non-malleability together with trusted binding tightly implies strong non-malleability. So, in this work we focus on obtaining the former two properties individually.

Theorem 5.4. *For any $t, Q, \epsilon', \epsilon'' \geq 0$ defining $\epsilon = \epsilon' + \epsilon''$, a VRF is (t, Q, ϵ)-strongly non-malleable (Definition 5.3) if it is $(t' = t, Q, \epsilon')$-non-malleable and $(t'' \approx t, Q, \epsilon'')$-trusted binding (Definition 5.1).*

Proof. Let $\mathcal{A}$ be any algorithm that attacks the strong non-malleability of the VRF, and has running time at most t and query complexity Q (to the prover). In the strong non-malleability experiment with $\mathcal{A}$, define two events as follows:

- NewProof: $\mathsf{Verify}_{pk}(\alpha^*, \pi^*) \neq \perp$ and π^* was not a response to any Prove_{sk} query;
- OldProof: $\mathsf{Verify}_{pk}(\alpha^*, \pi^*) \neq \perp$ and π^* was not a response to a $\mathsf{Prove}_{sk}(\alpha^*)$ query, but was a response to a $\mathsf{Prove}_{sk}(\alpha)$ query for some $\alpha \neq \alpha^*$.

Clearly, $\mathbf{Adv}^{\mathrm{SNM}}(\mathcal{A}) = \Pr[\mathsf{NewProof}] + \Pr[\mathsf{OldProof}]$.

Note that the only difference between the definitions of strong non-malleability and ordinary non-malleability is in the success condition, and the success event for non-malleability is exactly NewProof. Therefore,

$$\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) = \Pr[\mathsf{NewProof}].$$

We upper-bound $\Pr[\mathsf{OldProof}]$ via a straightforward reduction $\mathcal{R}$ that attacks the trusted binding of the VRF: $\mathcal{R}(pk)$ runs $\mathcal{A}(pk)$, and answers $\mathcal{A}$'s Prove_{sk} queries by forwarding them to its own Prove_{sk} oracle and relaying the answers (and similarly for any other oracles that $\mathcal{R}$ may have and $\mathcal{A}$ may expect to have). When $\mathcal{A}$ outputs some (α^*, π^*), let A be the set of $\mathcal{A}$'s queries to Prove_{sk} whose answer was π^*. If $A = \emptyset$ or $\alpha^* \in A$, then $\mathcal{R}$ aborts. Otherwise, $\mathcal{R}$ chooses an arbitrary $\alpha \in A$ and outputs $(\alpha, \alpha^*, \pi^*)$.

Clearly, $\mathcal{R}$'s running time is approximately $\mathcal{A}$'s, and $\mathcal{R}$'s query complexity is the same as $\mathcal{A}$'s. Now suppose that OldProof happens in the experiment that $\mathcal{R}$ runs. Then $\mathsf{Verify}_{pk}(\alpha^*, \pi^*) \neq \perp$ by definition of OldProof; $\mathsf{Verify}_{pk}(\alpha, \pi^*) \neq \perp$ because $\alpha \in A$ so (α, π^*) was a query-response pair of Prove_{sk}, which is valid by perfect completeness; and $\alpha^* \neq \alpha$ because $\alpha^* \notin A$ but $\alpha \in A$. We conclude that $\mathcal{R}$ succeeds in the trusted binding attack experiment whenever OldProof happens, so

$$\mathbf{Adv}^{\text{t-bind}}(\mathcal{R}) \geq \Pr[\mathsf{OldProof}].$$

Summing up, we get

$$\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) + \mathbf{Adv}^{\text{t-bind}}(\mathcal{R}) \geq \mathbf{Adv}^{\mathrm{SNM}}(\mathcal{A}),$$

and since $\mathcal{R}$'s running time is approximately $\mathcal{A}$'s, the claim follows. □

5.2 Full Binding

Here we show that ECVRF has full binding (even against quantum attacks) if the two hash functions HTC, H are collision resistant. This hypothesis holds if the functions are modeled as random oracles (even quantumly accessible ones) with sufficiently large output, as they are elsewhere, but for the present purposes only collision resistance is needed. The theorem follows straightforwardly from the fact that in a valid proof, the challenge c is a collision-resistant function of the public key and function input. (In the context of signature schemes, essentially the same observation was made in [CDF+21].) The theorem and its proof also adapt straightforwardly to other definitions of c, like those given in [PWH+17] and other versions of [GRPV22].

Theorem 5.5. *There exist adversaries $\mathcal{R}_{\mathsf{HTC}}$ and $\mathcal{R}_{\mathsf{H}}$ (explicitly given in the proof), attacking the collision resistance (Definition 2.2) of* HTC *and* H *respectively, such that for any (possibly quantum) adversary $\mathcal{A}$ attacking the full binding (Definition 5.2) of ECVRF (Algorithm 1), we have that*

$$\mathbf{Adv}^{CR}(\mathcal{R}^{\mathcal{A}}_{\mathsf{HTC}}) + \mathbf{Adv}^{CR}(\mathcal{R}^{\mathcal{A}}_{\mathsf{H}}) \geq \mathbf{Adv}^{\textit{f-bind}}(\mathcal{A}),$$

where $\mathcal{R}^{\mathcal{A}}_{\mathsf{HTC}}$ and $\mathcal{R}^{\mathcal{A}}_{\mathsf{H}}$ use oracle access to $\mathcal{A}$, and each of their total running times is approximately the running time of $\mathcal{A}$.

In particular, for any $t, \epsilon', \epsilon'' \geq 0$ defining $\epsilon = \epsilon' + \epsilon''$, ECVRF is (t, ϵ)-full binding if HTC *is $(t' \approx t, \epsilon')$-collision resistant and* H *is $(t'' \approx t, \epsilon'')$-collision resistant.*

Proof. Let $(X_0^*, \alpha_0^*, X_1^*, \alpha_1^*, \pi^*)$ denote $\mathcal{A}()$'s output in the full binding experiment. If $X_0^*, X_1^* \in \mathbb{E}$ and $\alpha_0^*, \alpha_1^* \in \mathcal{X}$ (which must be the case if $\mathcal{A}$ succeeds), let $H_0 = \mathsf{HTC}(X_0^*, \alpha_0^*)$ and $H_1 = \mathsf{HTC}(X_1^*, \alpha_1^*)$. Define HTCCol to be the event that $\mathcal{A}$ succeeds and $H_0 = H_1$, and HCol to be the event that $\mathcal{A}$ succeeds and $H_0 \neq H_1$. Clearly, $\mathbf{Adv}^{\text{f-bind}}(\mathcal{A}) = \Pr[\mathsf{HTCCol}] + \Pr[\mathsf{HCol}]$.

Define $\mathcal{R}_{\mathsf{HTC}} = \mathcal{R}_{\mathsf{HTC}}^{\mathcal{A}}$ to be the algorithm that runs $(X_0^*, \alpha_0^*, X_1^*, \alpha_1^*, \pi^*) \leftarrow \mathcal{A}()$ and outputs $((X_0^*, \alpha_0^*), (X_1^*, \alpha_1^*))$. Clearly, $\mathcal{R}_{\mathsf{HTC}}$'s total running time is approximately $\mathcal{A}$'s. If event HTCCol happens, then $(X_0^*, \alpha_0^*) \neq (X_1^*, \alpha_1^*)$ because $\mathcal{A}$ succeeds, and $\mathsf{HTC}(X_0^*, \alpha_0^*) = H_0 = H_1 = \mathsf{HTC}(X_1^*, \alpha_1^*)$. Therefore,

$$\mathbf{Adv}^{\text{CR}}(\mathcal{R}_{\mathsf{HTC}}) \geq \Pr[\mathsf{HTCCol}].$$

Next, define $\mathcal{R}_{\mathsf{H}} = \mathcal{R}_{\mathsf{H}}^{\mathcal{A}}$ to be the algorithm that:

- runs $(X_0^*, \alpha_0^*, X_1^*, \alpha_1^*, \pi^*) \leftarrow \mathcal{A}()$ and parses π^* as $(Z^* \in \mathbb{E}, c^* \in \mathbb{Z}_q, s^* \in \mathbb{Z}_q)$ (aborting otherwise),
- for each $b \in \{0, 1\}$, computes H_b as above and $R_{B,b} := B^{s^*}(X_b^*)^{-c^*}$, $R_{H,b} := (H_b)^{s^*}(Z^*)^{-c^*}$, and
- outputs $((H_0, Z^*, R_{B,0}, R_{H,0}), (H_1, Z^*, R_{B,1}, R_{H,1}))$.

Clearly, $\mathcal{R}_{\mathsf{H}}$'s total running time is approximately $\mathcal{A}$'s.

If HCol happens, then $(H_0, Z^*, R_{B,0}, R_{H,0}) \neq (H_1, Z^*, R_{B,1}, R_{H,1})$ because $H_0 \neq H_1$, and

$$\mathsf{H}(H_0, Z^*, R_{B,0}, R_{H,0}) = c^* = \mathsf{H}(H_1, Z^*, R_{B,1}, R_{H,1})$$

because $\mathcal{A}$ succeeds and thus both $\mathsf{Verify}_{X_0^*}(\alpha_0^*, \pi^*), \mathsf{Verify}_{X_1^*}(\alpha_1^*, \pi^*) \neq \perp$. Therefore,

$$\mathbf{Adv}^{\text{CR}}(\mathcal{R}_{\mathsf{H}}) \geq \Pr[\mathsf{HCol}].$$

The theorem follows by summing the advantages of $\mathcal{R}_{\mathsf{HTC}}$ and $\mathcal{R}_{\mathsf{H}}$. □

5.3 Non-Malleability

In this section we establish the following non-malleability theorem for ECVRF.

Theorem 5.6. *For any t and $Q = (Q_{\mathsf{P}}, Q_{\mathsf{HTC}}, Q_{\mathsf{H}})$, ECVRF (Algorithm 1) is (t, Q, ϵ)-non-malleable (Definition 5.3) for random oracles HTC and H, as long as the discrete logarithm problem in group $(\mathbb{G}, q, B)$ is (t', ϵ')-hard (Definition 2.1), where*

$$t' \approx 2t \quad \textit{and} \quad \epsilon' = \frac{\epsilon^2}{Q_{\mathsf{P}} + Q_{\mathsf{H}} + 1} - \left(\frac{2Q_{\mathsf{P}}}{|\mathbb{G}|} + \frac{1}{|\mathbf{H}|}\right)\epsilon.$$

5.3.1 Forking Lemma

Our proof of Theorem 5.6, given below in Section 5.3.2, uses the *forking lemma*. The original forking lemma of Pointcheval and Stern [PS96] is syntactically limited to signature schemes, and thus does not directly apply to our setting. Instead, we use the general forking lemma of Bellare and Neven [BN06].

Lemma 5.7 (General Forking Lemma). *Let $Q \in \mathbb{N}$ and $\mathbf{H}$ be a finite set. Let $\mathcal{W}$ be any randomized algorithm that on input X and $c_1, \ldots, c_Q \in \mathbf{H}$, outputs an integer $i \in \{0, \ldots, Q\}$ and a side output σ, and let IG be an input-generator algorithm for $\mathcal{W}$. Define $\mathcal{W}$'s* accepting probability *as*

$$\mathbf{Acc}(\mathcal{W}) := \Pr\left[i \neq 0 : \begin{array}{c} X \leftarrow \mathsf{IG}() \\ c_1, \ldots, c_Q \leftarrow \mathbf{H} \\ (i, \sigma) \leftarrow \mathcal{W}(X, c_1, \ldots, c_Q) \end{array}\right].$$

Consider the forking algorithm $\mathcal{F}^{\mathcal{W}}(X)$ *(Algorithm 5) that chooses and fixes $\mathcal{W}$'s random tape, runs $\mathcal{W}$ on X and random inputs from $\mathbf{H}$, and upon receiving $\mathcal{W}$'s output i, re-runs $\mathcal{W}$ on X and the same inputs from $\mathbf{H}$ but re-randomized from index i. The forking succeeds (i.e., $\mathcal{F}^{\mathcal{W}}(X)$ outputs 1 in the first output component) if $\mathcal{W}$ outputs the same index i in both runs, and the two sequences of random inputs differ at index i. Then*

$$\mathbf{Frk}(\mathcal{F}) := \Pr\left[b = 1 : \begin{array}{c} X \leftarrow \mathsf{IG}() \\ (b, \cdot, \cdot) \leftarrow \mathcal{F}^{\mathcal{W}}(X) \end{array}\right] \geq \mathbf{Acc}(\mathcal{W}) \cdot \left(\frac{\mathbf{Acc}(\mathcal{W})}{Q} - \frac{1}{|\mathbf{H}|}\right).$$

Algorithm 5 Forking algorithm $\mathcal{F}^{\mathcal{W}}(X)$

1: Choose a random tape ρ for $\mathcal{W}$ uniformly at random.
2: $c_1, \ldots, c_Q \leftarrow \mathbf{H}$
3: $(i, \sigma) := \mathcal{W}(X, c_1, \ldots, c_Q; \rho)$
4: **if** $i = 0$ **then return** $(0, \perp, \perp)$
5: $c'_i, \ldots, c'_Q \leftarrow \mathbf{H}$
6: $(i', \sigma') := \mathcal{W}(X, c_1, \ldots, c_{i-1}, c'_i, \ldots, c'_Q; \rho)$
7: **if** $i' = i \wedge c_i \neq c'_i$ **then return** $(1, \sigma, \sigma')$ **else return** $(0, \perp, \perp)$

5.3.2 Proof of Non-Malleability

Proof of Theorem 5.6. Let $\mathcal{A}$ be any algorithm that attacks the non-malleability of ECVRF, runs in time at most t, and makes at most Q_{H} queries to H and at most Q_{P} queries to $\mathsf{Prove}_{sk}(\cdot)$. Without loss of generality, we assume the following about $\mathcal{A}(X)$'s behavior:

- it never repeats a query to the deterministic HTC and H oracles (though it may repeat queries to Prove_{sk}, since the proving algorithm may be randomized);
- it queries $\mathsf{HTC}(X, \alpha)$ sometime before querying $\mathsf{Prove}_{sk}(\alpha)$; and
- sometime before it ultimately outputs a (not necessarily valid) input-proof pair (α^*, π^*), it queries $\mathsf{HTC}(X, \alpha^*)$ and also makes the associated H-query as in $\mathsf{Verify}_X(\alpha^*, \pi^*)$.[15]

Note that this final assumption may have the effect of implicitly increasing $\mathcal{A}$'s number of H-queries by one. Accordingly, let $Q' = Q_{\mathsf{P}} + Q_{\mathsf{H}} + 1$. (The number of HTC-queries does not play any role in the security bounds.)

We give a reduction $\mathcal{R}$ that uses $\mathcal{A}$ to attack the discrete logarithm problem in $(\mathbb{G}, q, B)$. To do this, we first define a wrapper algorithm $\mathcal{W}^{\mathcal{A}}$ (Algorithm 6) that, given a sequence of values $c_i \in \mathbf{H}$ representing H-outputs, simulates answers to all of $\mathcal{A}$'s oracle queries in the non-malleability experiment and, whenever $\mathcal{A}$ succeeds, identifies the H-query corresponding to $\mathcal{A}$'s output. We then use the generalized forking lemma (Lemma 5.7) with this wrapper algorithm to lower-bound the probability that forking succeeds in terms of $\mathcal{A}$'s advantage. Finally, we show that whenever forking succeeds for a given public key X, we can efficiently compute the discrete logarithm of X.

In the wrapper algorithm $\mathcal{W}^{\mathcal{A}}$, observe that the function H can be defined at specific inputs in two different ways: (1) when $\mathcal{A}$ explicitly queries H on an input for which H is not yet defined, on Line 9; and (2) when $\mathcal{W}$ simulates the proof oracle, on Line 17. We refer to the former type as 'explicit' definitions, and the latter type

[15] More specifically, for any public key X and any (not necessarily valid) input-proof pair $(\alpha \in \mathcal{X}, \pi = (Z \in \mathbb{E}, c \in \mathbb{Z}_q, s \in \mathbb{Z}_q))$, there is an associated H-input (H, Z, R_B, R_H), where $H = \mathsf{HTC}(X, \alpha) \in \mathbb{G}$, $R_B = B^s X^{-c} \in \mathbb{E}$, and $R_H = H^s Z^{-c} \in \mathbb{E}$.

Algorithm 6 Wrapper $\mathcal{W}^{\mathcal{A}}(X, c_1, \ldots, c_{Q'})$

1: $\mathsf{ctr} := 0$; $S := \emptyset$; $P := \emptyset$; initialize an empty table $T[1 \ldots Q']$
2: **run** $\mathcal{A}(X)$, answering $\mathcal{A}$'s oracle queries as follows:
3: **on query** $\mathsf{HTC}(X', \alpha)$: ▷ no repeated queries
4: $h \leftarrow \mathbb{Z}_q$; define $\mathsf{HTC}(X', \alpha) := B^h$
5: **if** $X' = X$ **then** record $[\alpha, h]$ ▷ record $h = \log_B(\mathsf{HTC}(X, \alpha))$
6: **return** $\mathsf{HTC}(X', \alpha)$ to $\mathcal{A}$
7: **on query** $\mathsf{H}(H, Z, R_B, R_H)$: ▷ no repeated queries. . .
8: **if** $\mathsf{H}(H, Z, R_B, R_H)$ is not defined **then** ▷ . . . but may have been defined on Line 17
9: $\mathsf{ctr} := \mathsf{ctr} + 1$; define $\mathsf{H}(H, Z, R_B, R_H) := c_{\mathsf{ctr}}$
10: define $T[\mathsf{ctr}] := (H, Z, R_B)$
11: **return** $\mathsf{H}(H, Z, R_B, R_H)$ to $\mathcal{A}$
12: **on query** $\mathsf{Prove}_{sk}(\alpha)$: ▷ $\mathsf{HTC}(X, \alpha)$ has been queried
13: $\mathsf{ctr} := \mathsf{ctr} + 1$; $S := S \cup \{\alpha\}$; $s \leftarrow \mathbb{Z}_q$ ▷ fresh s for every query
14: lookup $[\alpha, h]$; $H := B^h$; $Z := X^h$ ▷ simulate $Z = H^x = B^{hx}$ as X^h
15: $R_B := B^s X^{-c_{\mathsf{ctr}}}$; $R_H := H^s Z^{-c_{\mathsf{ctr}}}$
16: **if** $\mathsf{H}(H, Z, R_B, R_H)$ is defined **then return** $(0, \perp)$ ▷ hash can't be two different values
17: define $\mathsf{H}(H, Z, R_B, R_H) := c_{\mathsf{ctr}}$
18: $\pi := (Z, c_{\mathsf{ctr}}, s)$; $P := P \cup \{\pi\}$ ▷ construct and store proof
19: **return** π to $\mathcal{A}$
20: When $\mathcal{A}$ outputs $(\alpha^* \in \mathcal{X}, \pi^* = (Z^* \in \mathbb{E}, c^* \in \mathbb{Z}_q, s^* \in \mathbb{Z}_q))$: ▷ $\mathsf{HTC}(X, \alpha^*)$ has been queried
21: $H^* := \mathsf{HTC}(X, \alpha^*)$; $R_B^* := B^{s^*} X^{-c^*}$; $R_H^* := (H^*)^{s^*} (Z^*)^{-c^*}$
▷ $\mathsf{H}(H^*, Z^*, R_B^*, R_H^*)$ has been queried
22: **if** $\mathsf{Verify}_X(\alpha^*, \pi^*) = \perp \vee \pi^* \in P$ **then return** $(0, \perp)$ ▷ $\mathcal{A}$ fails
23: Find some $i \in \{1, \ldots, Q'\}$ such that $T[i] = (H^*, Z^*, R_B^*)$ ▷ By Claim 5.8, such i exists
24: **return** $(i, (c^*, s^*))$

as 'implicit' definitions. For each definition (no matter which type), $\mathcal{W}$ answers with the next c_{ctr} value from its input sequence. Thus, at most Q' values of c_{ctr} are needed by $\mathcal{W}^{\mathcal{A}}$.

We now justify the second-to-last line of the wrapper algorithm.

Claim 5.8. If Line 23 of Algorithm 6 is reached then it succeeds, i.e., there exists an $i \in \{1, \ldots, Q'\}$ such that $T[i] = (H^*, Z^*, R_B^*)$.

Proof. Suppose that Line 23 is reached. Then $\mathsf{H}(H^*, Z^*, R_B^*, R_H^*)$ has been defined, because at some point it was queried by our assumption on $\mathcal{A}$ (see the comment on Line 21). We show that it was defined in an *explicit* (not an implicit) definition, hence such i exists due to Line 10.

Recall that $\mathcal{A}$ makes Q_{P} queries to Prove_{sk}, and say that for each such query it receives some answer (Z_ℓ, c_ℓ, s_ℓ) (for some $\ell \in \{1, \ldots, Q'\}$, where c_ℓ is one of $\mathcal{W}^{\mathcal{A}}$'s inputs). This answer has the associated implicit H-definition for the input tuple

$$(H_\ell, Z_\ell, R_{B,\ell} = B^{s_\ell} X^{-c_\ell}, R_{H,\ell} = H_\ell^{s_\ell} Z_\ell^{-c_\ell}).$$

We show that (H^*, Z^*, R_B^*, R_H^*) is not among these tuples.

Suppose for contradiction that $(H^*, Z^*, R_B^*, R_H^*) = (H_\ell, Z_\ell, R_{B,\ell}, R_{H,\ell})$ for some ℓ. Then $Z^* = Z_\ell$, and

$$c^* = \mathsf{H}(H^*, Z^*, R_B^*, R_H^*) = \mathsf{H}(H_\ell, Z_\ell, R_{B,\ell}, R_{H,\ell}) = c_\ell,$$

where the first equality holds because $\mathsf{Verify}_X(\alpha^*, \pi^*) \neq \perp$ (see Line 22); the second equality holds because H is a function (and in particular $\mathcal{W}^{\mathcal{A}}$ never aborted on Line 16); and the third equality holds by Line 17. Therefore,

$$B^{s^*} X^{-c^*} = R_B^* = R_{B,\ell} = B^{s_\ell} X^{-c_\ell},$$

and since $c^* = c_\ell$, we have $B^{s^*} = B^{s_\ell}$ and thus $s^* = s_\ell \in \mathbb{Z}_q$ (because B is a generator of $\mathbb{G}$). But then $(Z^*, c^*, s^*) = (Z_\ell, c_\ell, s_\ell)$, so $\pi^* = (Z^*, c^*, s^*) \in P$ was an answer from one of $\mathcal{A}$'s Prove_{sk} queries, contradicting the fact that Line 23 was reached (due to Line 22). □

Now let IG be the algorithm that lets $(X = B^x, x \in \mathbb{Z}_q) \leftarrow \mathsf{Gen}$ and outputs X only. The following claim shows that $\mathcal{W}$ perfectly simulates the non-malleability experiment to $\mathcal{A}$, as long as $\mathcal{W}$ does not abort on Line 16.

Claim 5.9. For $X \leftarrow \mathsf{IG}$ and $c_1, \ldots, c_{Q'} \leftarrow \mathbf{H}$, suppose that $\mathcal{W}(X, c_1, \ldots, c_{Q'})$ never aborts on Line 16. Then $\mathcal{A}$'s 'simulated' view while run as a subroutine of $\mathcal{W}$ is identically distributed to $\mathcal{A}$'s 'real' view in the non-malleability experiment (Definition 5.3) against ECVRF.

Proof. In both views, the public key $X = B^x \in \mathbb{G}$ for some uniformly random $x \leftarrow \mathbb{Z}_q \setminus \{0\}$. Moreover, the HTC oracle is implemented via lazy uniform sampling from $\mathbb{G}$, so $\mathcal{A}$'s views of these oracles are identically distributed. For the rest of the proof, fix some arbitrary $X \in \mathbb{G}$ (and its discrete log x) and HTC function.

We now consider H and Prove_{sk} queries (jointly). Recall that in the real view, the H oracle can equivalently be implemented by lazy sampling from $\mathbf{H}$ (as it is in the simulated view), and each $\mathsf{Prove}_{sk}(\alpha)$ query is answered by (Z, c, s), where $s = r + x \cdot c \in \mathbb{Z}_q$ for a fresh uniformly random $r \leftarrow \mathbb{Z}_q$, and $c = \mathsf{H}(H, Z, R_B, R_H)$ where $R_B = B^r, R_H = H^r$ (and H and Z are determined by α and the fixed X and HTC). We next describe three successive changes to the implementation of the Prove_{sk} oracle that leave the distribution of $\mathcal{A}$'s view unchanged, unless a certain event occurs.

1. First, instead of defining $c := \mathsf{H}(H, Z, R_B, R_H)$, we first choose a fresh uniformly random $c \leftarrow \mathbf{H}$ and then define $\mathsf{H}(H, Z, R_B, R_H) := c$ *as long as* $\mathsf{H}(H, Z, R_B, R_H)$ has not already been defined, in which case the experiment aborts; this new behavior corresponds to Lines 16 and 17 of the simulation.

2. Second, instead of choosing uniform r, we choose a fresh uniform $s \leftarrow \mathbb{Z}_q$ and set $r := s - x \cdot c \in \mathbb{Z}_q$, $R_B := B^{s-x\cdot c} = B^s(B^x)^{-c} = B^s X^{-c}$ and similarly $R_H := H^s Z^{-c}$, as is done in the simulation.

3. Third, since $H^x = B^{hx} = X^h$ (where $h = \log_B(\mathsf{HTC}(X, \alpha))$), instead of defining $Z := H^x$ as in the real view, we define $Z := X^h$ as is done in the simulation.

With all these changes, the resulting view is exactly the simulated one provided by $\mathcal{W}$, as desired. □

We now analyze the probability that $\mathcal{W}$ ever aborts on Line 16. This event can happen only while preparing the answer to a Prove_{sk} query. For each such query, $s \leftarrow \mathbb{Z}_q$ is a fresh uniformly random exponent, so the corresponding $R_B = B^s X^{-c_{\mathrm{ctr}}} \in \mathbb{G}$ is uniformly random and independent of all the other random variables. Note that $\mathcal{W}$ defines H on at most Q' input tuples. Thus, the probability that R_B is the third component of one of these tuples (which is necessary for $\mathcal{W}$ to abort on Line 16) is at most $Q'/|\mathbb{G}|$. So, by the union bound, we have that

$$\Pr[\mathcal{W} \text{ ever aborts on Line 16}] \le \frac{Q_{\mathsf{P}} \cdot Q'}{|\mathbb{G}|}.$$

Because Line 23 is reached exactly when $\mathcal{A}$ succeeds in the simulated non-malleability experiment, and recalling the definition of $\mathbf{Acc}(\mathcal{W})$ from Lemma 5.7, by Claim 5.9 we have

$$\mathbf{Acc}(\mathcal{W}) \ge \mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) - \Pr[\mathcal{W} \text{ ever aborts on Line 16}] \ge \mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) - \frac{Q_{\mathsf{P}} \cdot Q'}{|\mathbb{G}|}. \tag{5.4}$$

Let $\mathcal{F} = \mathcal{F}^{\mathcal{W}}$ be the forking algorithm (see Lemma 5.7) associated with $\mathcal{W}$. Then by the forking lemma,

$$\mathbf{Frk}(\mathcal{F}) \ge \mathbf{Acc}(\mathcal{W}) \left(\frac{\mathbf{Acc}(\mathcal{W})}{Q'} - \frac{1}{|\mathbf{H}|} \right).$$

Algorithm 7 Reduction $\mathcal{R}(X)$

$(b, \sigma, \sigma') \leftarrow \mathcal{F}(X)$
if $(b, \sigma, \sigma') = (0, \bot, \bot)$ **then return** $\bot$
parse $\sigma = (c, s)$ and $\sigma' = (c', s')$; **return** $x = (s - s')(c - c')^{-1} \in \mathbb{Z}_q$ ▷ $c \neq c'$ by construction of $\mathcal{F}$

We use $\mathcal{F}$ to construct a reduction $\mathcal{R}$ (Algorithm 7) that solves the discrete logarithm problem in $(\mathbb{G}, q, B)$. Specifically, whenever the forking algorithm $\mathcal{F}(X)$ succeeds inside $\mathcal{R}(X)$, the latter finds the discrete logarithm of X.

Claim 5.10. Suppose $\mathcal{F}(X)$ outputs $(1, \sigma = (c, s), \sigma' = (c', s'))$. Then $B^s X^{-c} = B^{s'} X^{-c'}$ and $c \neq c'$, thus $\mathcal{R}$'s output $x \in \mathbb{Z}_q$ satisfies $B^x = X$.

Proof. By definition of $\mathcal{F}(X)$, if it outputs $(1, \sigma = (c, s), \sigma' = (c', s'))$, then there exists an index $i \in \{1, \ldots, Q'\}$ and random tape ρ such that $c_i \neq c'_i$, and $\mathcal{W}$'s two executions $\mathcal{W}(X, c_1, \ldots, c_{Q'}; \rho)$ and $\mathcal{W}(X, c_1, \ldots, c_{i-1}, c'_i, \ldots, c'_{Q'}; \rho)$ output $(i, (c, s))$ and $(i, (c', s'))$, respectively.

By definition, this implies that in $\mathcal{W}$'s two executions, $\mathcal{A}$ respectively outputs valid input-proof pairs $(\alpha, \pi = (Z, c, s))$ and $(\alpha', \pi' = (Z', c', s'))$ that cause $\mathcal{W}$ to reach Line 23. That is, letting $H = \mathsf{HTC}(X, \alpha)$, $R_B = B^s X^{-c}$, and R_H be as in $\mathsf{Verify}_X(\alpha, \pi)$ from the first execution, we have $c = \mathsf{H}(H, Z, R_B, R_H) = c_i$, where the first equality holds because $\mathsf{Verify}_X(\alpha, \pi) \neq \perp$, and the second equality holds by Line 23. Similarly for the second execution, let $H' = \mathsf{HTC}(X, \alpha')$, $R'_B = B^{s'} X^{-c'}$, and R'_H be as in $\mathsf{Verify}_X(\alpha', \pi')$, so we have $c' = \mathsf{H}(H', Z', R'_B, R'_H) = c'_i$.

Now observe that $\mathcal{W}$'s two executions are *identical* until they use c_i or c'_i, respectively. By Line 23, $\mathsf{H}(H, Z, R_B, R_H)$ (resp., $\mathsf{H}(H', Z', R'_B, R'_H)$) is defined *explicitly*, so c_i (resp., c'_i) is not used until after $\mathcal{A}$ explicitly makes this H-query. Therefore, the two queries (H, Z, R_B, R_H) and (H', Z', R'_B, R'_H) must be equal.[16] It follows that $B^s X^{-c} = R_B = R'_B = B^{s'} X^{-c'}$, so $X^{c-c'} = B^{s-s'}$, and the claim follows. □

Finally, by all of the above, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{DL}}(\mathcal{R}) &= \mathbf{Frk}(\mathcal{F}) \\
&\geq \mathbf{Acc}(\mathcal{W}) \left(\frac{\mathbf{Acc}(\mathcal{W})}{Q'} - \frac{1}{|\mathbf{H}|} \right) \\
&\geq \left(\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) - \frac{Q_{\mathsf{P}} \cdot Q'}{|\mathbb{G}|} \right) \left(\frac{\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) - Q_{\mathsf{P}} \cdot Q' / |\mathbb{G}|}{Q'} - \frac{1}{|\mathbf{H}|} \right) \\
&= \left(\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}) - \frac{Q_{\mathsf{P}} \cdot Q'}{|\mathbb{G}|} \right) \left(\frac{\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A})}{Q'} - \frac{Q_{\mathsf{P}}}{|\mathbb{G}|} - \frac{1}{|\mathbf{H}|} \right) \\
&\geq \frac{\mathbf{Adv}^{\mathrm{NM}}(\mathcal{A})^2}{Q'} - \left(\frac{2Q_{\mathsf{P}}}{|\mathbb{G}|} + \frac{1}{|\mathbf{H}|} \right) \mathbf{Adv}^{\mathrm{NM}}(\mathcal{A}).
\end{aligned}$$

Finally, the running time of $\mathcal{R}$ is approximately that of $\mathcal{F}$, which is approximately twice that of $\mathcal{W}$ and $\mathcal{A}$. This completes the proof of Theorem 5.6. □

Remark 5.11. We note that nothing in the proof of Theorem 5.6 uses the fact that H and R_H are part of the input to H. However, the inclusion of H was used in Section 5.2 to establish full binding, and the inclusion of R_H is needed for the soundness of the Σ-protocol that underlies the uniqueness of ECVRF (see Section 4).

References

[AMNR18] I. Abraham, D. Malkhi, K. Nayak, and L. Ren. Dfinity consensus, explored. Cryptology ePrint Archive, Paper 2018/1153, 2018. https://eprint.iacr.org/2018/1153. Page 2.

[BCJZ21] J. Brendel, C. Cremers, D. Jackson, and M. Zhao. The provable security of Ed25519: Theory and practice. In *IEEE Symposium on Security and Privacy*, pages 1659–1676. 2021. Pages 3 and 26.

[BCKL09] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. Compact e-cash and simulatable VRFs revisited. In *Pairing*, pages 114–131. 2009. Page 2.

[16] We emphasize that this reasoning would not hold if H were *implicitly* defined on these inputs, because $\mathcal{W}$'s simulation of Prove_{sk} uses c_{ctr} to construct the input at which it defines H. So, the two hash inputs would not necessarily be equal between the two executions. This reasoning is where we rely on the fact that $\mathcal{A}$ outputs a *new* proof that it did not receive from its Prove_{sk} oracle, which by Claim 5.8 corresponds to an explicit definition.

[BDF+11] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random oracles in a quantum world. In *ASIACRYPT*, pages 41–69. 2011. Pages 2 and 6.

[BGK+18] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. 2018. Page 2.

[BN06] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, pages 390–399. 2006. Pages 3 and 29.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. 1993. Pages 4, 6, 17, and 18.

[CDF+21] C. Cremers, S. Düzlü, R. Fiedler, M. Fischlin, and C. Janson. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. In *IEEE Symposium on Security and Privacy*, pages 1696–1714. 2021. Pages 3, 26, and 28.

[CDMP05] J. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In *CRYPTO*, pages 430–448. 2005. Pages 4, 5, 13, and 16.

[CGN20] K. Chalkias, F. Garillot, and V. Nikolaenko. Taming the many EdDSAs. In *Security Standardisation Research*, pages 67–90. 2020. Pages 3 and 26.

[CM19] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. Page 2.

[CP92] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105. 1992. Pages 4 and 17.

[DFM20] J. Don, S. Fehr, and C. Majenz. The measure-and-reprogram technique 2.0: Multi-round Fiat-Shamir and more. In *CRYPTO*, pages 602–631. 2020. Page 18.

[DFMS19] J. Don, S. Fehr, C. Majenz, and C. Schaffner. Security of the Fiat–Shamir transformation in the quantum random-oracle model. In *CRYPTO*, pages 356–383. 2019. Pages 4, 17, 18, and 19.

[DGKR18] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT*, pages 66–98. 2018. Page 2.

[ESLR22] M. F. Esgin, R. Steinfeld, D. Liu, and S. Ruj. Efficient hybrid exact/relaxed lattice proofs and applications to rounding and VRFs. Cryptology ePrint Archive, Report 2022/141, 2022. https://eprint.iacr.org/2022/141. Page 5.

[FHSS+22] A. Faz-Hernández, S. Scott, N. Sullivan, R. S. Wahby, and C. A. Wood. Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve, IETF Secretariat, 2022. Working Draft, https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve. Page 8.

[FPS20] G. Fuchsbauer, A. Plouviez, and Y. Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In *EUROCRYPT*, pages 63–95. 2020. Page 5.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194. 1986. Pages 4, 17, and 18.

[GHM+17] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. 2017. Page 2.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. 1996. Page 13.

[GRPV22] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf, IETF Secretariat, 2022. Working Draft, https://datatracker.ietf.org/doc/draft-irtf-cfrg-vrf. Pages 2, 3, 4, 6, 7, 8, 9, and 28.

[MR01] S. Micali and L. Reyzin. Soundness in the public-key model. In *CRYPTO*, pages 542–565. 2001. Page 2.

[MR02] S. Micali and R. L. Rivest. Micropayments revisited. In *CT-RSA*, pages 149–163. 2002. Page 2.

[MRH04] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, pages 21–39. 2004. Pages 4 and 9.

[MRV99] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. 1999. Pages 1 and 8.

[PS96] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, pages 387–398. 1996. Page 29.

[PWH+17] D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Včelák, L. Reyzin, and S. Goldberg. Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099, 2017. https://eprint.iacr.org/2017/099. Pages 2, 8, and 28.

[RS21] L. Rotem and G. Segev. Tighter security for Schnorr identification and signatures: A high-moment forking lemma for σ-protocols. In *CRYPTO*, pages 222–250. 2021. Page 5.

[Sch89] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991. Preliminary version in CRYPTO 1989. Page 3.

[Sho94] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. Preliminary version in FOCS 1994. Page 2.

[Unr17] D. Unruh. Post-quantum security of Fiat-Shamir. In *ASIACRYPT*, pages 65–95. 2017. Pages 4, 18, and 19.

[Unr21] D. Unruh. Compressed permutation oracles (and the collision-resistance of sponge/SHA3). Cryptology ePrint Archive, Paper 2021/062, 2021. https://eprint.iacr.org/2021/062. Page 12.

[VGP+18] J. Včelák, S. Goldberg, D. Papadopoulos, S. Huque, and D. C. Lawrence. NSEC5, DNSSEC Authenticated Denial of Existence. Internet-Draft draft-vcelak-nsec5, IETF Secretariat, 2018. Working Draft, https://datatracker.ietf.org/doc/draft-vcelak-nsec5/. Page 2.

[Zha19] M. Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In *CRYPTO*, pages 239–268. 2019. Pages 4, 5, 9, 10, 11, 12, 13, 16, 17, 22, 37, 38, and 39.

A Proofs for Section 3

Here we recall and prove the unproven lemmas from Section 3.

A.1 Quantum Indistinguishability

Lemma 3.19. *For any $\tilde{Q} = (\tilde{Q}_{D,1}, \tilde{Q}_{D,2})$, the simulator $\mathcal{S}$ from the proof of Theorem 3.18 is $(\tilde{Q}, \epsilon)$-indistinguishable (Definition 3.3), where*

$$\epsilon = O(\tilde{Q}_{D,1} \cdot (\tilde{Q}_{D,1}^{1/2} + \tilde{Q}_{D,2})/\sqrt{|\mathcal{Y}_1|}).$$

Proof. Following the definition of indistinguishability, consider a distinguisher $\widetilde{\mathcal{D}}$ with query complexity $(\tilde{Q}_{D,1}, \tilde{Q}_{D,2})$ that is given quantum access either to oracles O, H_2 (the 'real' experiment) or to oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}$ (the 'ideal' experiment).

The rest of the proof closely parallels the one for [Zha19, Lemma 8], which shows ordinary indistinguishability of its simulator (not relative to any oracle). Therefore, we mainly describe the differences between the proofs, i.e., their hybrid experiments and justifications for why adjacent hybrids are distinguishable with only bounded advantage. Overall, the hybrids are the same as in [Zha19], except that the distinguisher additionally has access to $\mathsf{FI}_{\mathsf{H}_1}$ in all of them.

Hybrid 0 is the 'real' experiment, where the distinguisher $\widetilde{\mathcal{D}}$ has oracles $\mathsf{O} = (\mathsf{H}_1, \mathsf{FI}_{\mathsf{H}_1})$ and H_2, where $\mathsf{H}_1, \mathsf{H}_2$ are implemented as compressed oracles with databases $\mathbf{D}_1, \mathbf{D}_2$ (respectively).

Hybrid 1 is the same, but with an abort condition: after each query to H_1, we measure if the database $\mathbf{D}_1$ contains a collision, and immediately abort the experiment if so. (Conceptually, we could also measure for a collision in $\mathbf{D}_1$ after each query to $\mathsf{FI}_{\mathsf{H}_1}$ or H_2, but because such queries do not affect $\mathbf{D}_1$, this would never detect a collision.)

Claim A.1. The difference between the probabilities that $\widetilde{\mathcal{D}}$ accepts in Hybrids 0 and 1 is $O(\tilde{Q}_{D,1}^{3/2}/\sqrt{|\mathcal{Y}_1|})$.

The only difference between the above hybrids and the corresponding ones defined in [Zha19] is that here the distinguisher additionally has oracle access to $\mathsf{FI}_{\mathsf{H}_1}$. By Lemma 3.12, this access cannot increase the collision probability for $\mathbf{D}_1$.

Hybrid 2 is defined analogously to the one in [Zha19]. There are now three databases $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}'$. Immediately before each query to H_1, H_2, *or* $\mathsf{FI}_{\mathsf{H}_1}$ (this addition is only difference with [Zha19]), these three databases are 'decoded' to just two databases $\mathbf{D}_1, \mathbf{D}_2$, then the query is handled using these databases, then they are immediately 'encoded' back to three databases.

Claim A.2. $\widetilde{\mathcal{D}}$ has the same acceptance probability in Hybrids 1 and 2.

This follows by the same reasoning as in [Zha19, Lemma 10]. Essentially, we can indistinguishably modify Hybrid 1 to introduce an 'encode' immediately followed by a 'decode' between every pair of adjacent queries (and also before the first one and after the last one). Each 'encode' then commutes with the collision check that is done after each H_1 query, yielding Hybrid 2.

Hybrid 3 also is analogous to the one in [Zha19]: it is the 'ideal' experiment (where H_2-queries are answered by $\mathcal{S}^{\mathsf{O},\mathsf{H}'}$) except that we still have the abort condition if a collision in $\mathbf{D}_1$ is ever measured after an H_1-query.

In other words, instead of handling queries by decoding the three databases to two, answering the query, and then encoding back to three databases, we act directly on the three databases as in the ideal experiment with $\mathcal{S}$. So, for H_1- and $\mathsf{FI}_{\mathsf{H}_1}$-queries, the difference with Hybrid 2 is just that the queries are made directly using $\mathbf{D}_1$ (without any decoding and encoding). For H_2-queries, due to our 'refactoring' of the simulator, the process ends up being exactly as described in [Zha19, Hybrid 3].

Claim A.3. The difference between $\widetilde{\mathcal{D}}$'s acceptance probabilities in Hybrids 2 and 3 is $O(\tilde{Q}_{D,1} \cdot \tilde{Q}_{D,2}/\sqrt{|\mathcal{Y}_1|})$.

This follows by reasoning very similar to that in the proof of [Zha19, Lemma 11], as summarized here. For each H_1-query, it suffices to swap the order of the corresponding 'encoding' and CStO operations, which by their almost-commutativity introduces $O(\tilde{Q}_{D,1} \cdot \tilde{Q}_{D,2}/\sqrt{|\mathcal{Y}_1|})$ total distinguishing advantage. (More specifically, each of the at most $\tilde{Q}_{D,1}$ CStO operations needs to be swapped with the at most $2\tilde{Q}_{D,2}$ $\mathsf{FindInput}$ operations performed by each 'encoding' done in superposition.) Similarly, for each $\mathsf{FI}_{\mathsf{H}_1}$-query, it suffices to swap the 'encoding' and $\mathsf{FindInput}$ operations; these commute (with perfect indistinguishability) because they interact only through the contents of the database $\mathbf{D}_1$, which they do not alter. Finally, the handling of H_2-queries in our Hybrids 2 and 3 is exactly as in [Zha19], and as shown in the proof of [Zha19, Lemma 11], this handling is identical in the two experiments.

Hybrid 4 is exactly the 'ideal' experiment, which has no abort-upon-collision condition.

Claim A.4. The difference between $\widetilde{\mathcal{D}}$'s acceptance probabilities in Hybrids 3 and 4 is $O(\tilde{Q}_{D,1}^{3/2}/\sqrt{|\mathcal{Y}_1|})$.

The only difference between Hybrids 3 and 4 is whether they abort upon a collision in $\mathbf{D}_1$, so the claim again follows by Lemma 3.12. This completes the proof of Lemma 3.19. □

A.2 Quantum Consistency

Lemma 3.20. *The simulator $\mathcal{S}$ from the proof of Theorem 3.18 is (Q_D, ϵ)-consistent (Definition 3.4), where Q_D is as in the statement and*

$$\epsilon = O((Q_{D,1} + Q_{D,3})^{3/2}/\sqrt{|\mathcal{Y}_1|}) = O(Q_D^{3/2}/\sqrt{|\mathcal{Y}_1|}).$$

Proof. Following the definition of consistency, consider a distinguisher $\widetilde{\mathcal{D}}$ with query complexity $Q_D = (Q_{D,1}, Q_{D,2}, Q_{D,3})$ that is given quantum access either to oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{C}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}}$ (the 'real' experiment) or to oracles $\mathsf{O}, \mathcal{S}^{\mathsf{O},\mathsf{H}'}, \mathsf{H}'$ (the 'ideal' experiment), where $\mathsf{H}' \colon \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}_2$ is a random oracle.

The rest of the proof closely parallels the one for [Zha19, Lemma 13], which shows ordinary consistency of its simulator (not relative to any oracle). Indeed, the differences are even less significant than those in the proof of Lemma 3.19 above, so we will be fairly brief. Overall, we use the same hybrid experiments as in [Zha19], except that the distinguisher additionally has access to $\mathsf{FI}_{\mathsf{H}_1}$ in all of them. This only affects the parts of the argument that use the compressed-oracle collision bound; for these we can use the FICO collision bound (Lemma 3.12) instead.

Hybrid 0 is the 'real' experiment, where the distinguisher $\widetilde{\mathcal{D}}$'s third oracle is $\mathsf{C}^{\mathsf{O},\mathcal{S}^{\mathsf{O},\mathsf{H}'}}$. Due to our 'refactoring' of the simulator, this oracle answers queries using the same process as appears at the start of the proof of [Zha19, Lemma 13]. So, this hybrid is the same as the one from [Zha19], but with the addition of the $\mathsf{FI}_{\mathsf{H}_1}$ interface.

Hybrid 1 is the same as the one from [Zha19], but with the addition of the $\mathsf{FI}_{\mathsf{H}_1}$ interface. (The only changes compared to Hybrid 0 are in the procedure used for answering queries to $\widetilde{\mathcal{D}}$'s third oracle.)

Claim A.5. The difference between $\widetilde{\mathcal{D}}$'s acceptance probabilities in Hybrids 0 and 1 is $O(Q_{D,3}/\sqrt{|\mathcal{Y}_1|})$.

This follows by the same reasoning as in [Zha19], using almost-commutativity (the additional $\mathsf{FI}_{\mathsf{H}_1}$ interface does not affect the argument).

Hybrid 2 is the same as the one from [Zha19], but with the addition of the $\mathsf{FI}_{\mathsf{H}_1}$ interface. Namely, after every query to H_1 we measure if $\mathbf{D}_1$ has a collision, and abort if so.

Claim A.6. The difference between $\widetilde{\mathcal{D}}$'s acceptance probabilities in Hybrids 1 and 2 is $O((Q_{D,1}+Q_{D,3})^{3/2}/\sqrt{|\mathcal{Y}_1|})$.

This follows by the same reasoning as in [Zha19], but instead using the FICO collision bound (Lemma 3.12).

Hybrids 3 and 4 are the same as the ones from [Zha19], but with the addition of the $\mathsf{FI}_{\mathsf{H}_1}$ interface. (The only changes compared to Hybrid 2 are in the procedure used for answering queries to $\widetilde{\mathcal{D}}$'s third oracle.)

Claim A.7. $\widetilde{\mathcal{D}}$ has the same acceptance probabilities in Hybrids 2, 3, and 4.

This follows by the same reasoning as in [Zha19], which is unaffected by the additional $\mathsf{FI}_{\mathsf{H}_1}$ interface.

Hybrid 5 simply removes the abort-upon-collision condition. This is the 'ideal' experiment.

Claim A.8. The difference between $\widetilde{\mathcal{D}}$'s acceptance probabilities in Hybrids 4 and 5 is $O(Q_{D,1}^{3/2}/\sqrt{|\mathcal{Y}_1|})$.

This again follows from the FICO collision bound (Lemma 3.12). Note that $\widetilde{\mathcal{D}}$'s (at most) $Q_{D,3}$ queries to the third oracle have no effect on this analysis because they do not induce any H_1-queries in these hybrids. This completes the proof. □