On the Non-Malleability of ECVRF in the Algebraic Group Model *

Willow Barkan-Vered, Franklin Harding, Jonathan Keller, and Jiayu Xu

Oregon State University, {barkanvt,hardingf,kellerjo,xujiay}@oregonstate.edu

Abstract. ECVRF is a verifiable random function (VRF) scheme used in multiple cryptocurrency systems. It has recently been proven to satisfy the notion of non-malleability which is useful in applications to blockchains (Peikert and Xu, CT-RSA 2023); however, the existing proof uses the rewinding technique and has a quadratic security loss. In this work, we re-analyze the non-malleability of ECVRF in the algebraic group model (AGM) and give a *tight* proof. We also compare our proof with the unforgeability proof for the Schnorr signature scheme in the AGM (Fuchsbauer, Plouviez and Seurin, EUROCRYPT 2020).

1 Introduction

Verifiable Random Functions (VRFs), proposed by Micali, Rabin and Vadhan [8], are a cryptographic primitive that allows a party who holds a secret key to compute a PRF output together with a proof, and anyone who holds the corresponding public key can verify that the output was computed correctly given the proof. The basic security properties are (1) *uniqueness*: for any public key and any input, it is infeasible to generate two output/proof pairs with different outputs but the verification algorithm accepts on both pairs; and (2) *pseudorandomness*: for anyone who only knows the public key, the output is indistinguishable from random.

In recent years, new applications for VRFs have been realized, especially in the cryptocurrency sector. This context requires some new security properties that have not been considered before. One of them is *non-malleability* [10]; at a high level, it means that given the public key and oracle access to the proving algorithm (but not the secret key), it is infeasible to generate a valid input/proof pair where the proof is not the output of a proof oracle query. This property might be useful in the following scenario: in a consensus protocol, if a malicious gossiper modifies the proof sent from an honest party, it might be difficult for parties in the network to reach consensus on the "right" proof. Non-malleability of VRFs is similar to unforgeability of signature schemes: if we view the proving

* This is the final report of CS 529 (Topics in Cryptography) at Oregon State University, a 10-week course taught by the last author in spring 2023. Its goal is to help students understand the basic principles of reading and writing papers in cryptography.

algorithm as a signing algorithm, then non-malleability says that it is infeasible to generate a forgery where the signature is new (as opposed to unforgeability where the message must be new).

Elliptic Curve VRF (ECVRF) [9] is a particularly efficient VRF scheme that has been used in multiple cryptocurrency systems such as Algorand [6] and Cardano [2], and is currently being considered for standardization [7]. It is based on the hardness of the discrete logarithm (DL) problem in an elliptic curve group, and its designing idea has some similarities to the Schnorr signature scheme [12]: roughly, the proving algorithm is done via applying the Fiat–Shamir heuristics [3] to the Chaum–Pedersen sigma protocol for proving equality of discrete logarithms [1] (whereas Schnorr signature is based on the Schnorr sigma protocol for proof of knowledge of discrete logarithm).

ECVRF has been proven non-malleable in the random oracle model (under the DL assumption) in a recent paper by Peikert and Xu [10]; as in the security proof for the Schnorr signature scheme [11], the ECVRF non-malleability proof in [10] uses the rewinding technique which incurs a quadratic security loss (see [10, Theorem 5.6]). A natural question is whether this loss can be avoided in some stronger model such as the algebraic group model (AGM) [4], as mentioned in [10, Section 1.2]:

> *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) [5] [...], analogous results for ECVRF's non-malleability seem plausible.*

Indeed, the recent paper by Fuchsbauer, Plouviez and Seurin [5] gives a tight unforgeability proof of the Schnorr signature scheme in the AGM. Recall that the Schnorr signature has the form (R, s), where R is a group element and s is an integer; at a very high level, the proof idea is to obtain two expressions of R using the group generator g and the public key X — one based on the adversary's winning condition and the other based on the fact that the adversary is algebraic — from which the reduction to DL can solve for $\log X$.

In this work, we solve the open problem mentioned in [10] and give a *tight* ECVRF non-malleability proof in the ROM+AGM. Our proof idea is similar to [5]; one main difference is that the second expression of R is obtained via observing the adversary's random oracle queries, as R is not part of the adversary's final output. Another difference is that upper-bounding the probability that the two expressions of R are identical (in which case the reduction cannot solve DL) is more complicated in our proof.

The rest of this paper is organized as follows. In Section 2 we review necessary concepts including the DL assumption, the Schnorr signature scheme and its security definition, and ECVRF and its non-malleability definition. Then as a warm-up, in Section 3 we rewrite the Schnorr signature security proof in [5]. The main reason why we do this is that the proof in [5] uses the "game-hop" methodology, while the ECVRF non-malleability proof in the ROM in [10] uses the more straightforward approach of presenting a reduction

and then analyzing its winning probability by ruling out "bad events"; thus, rewriting the former proof helps us compare our ECVRF non-malleability proof in the ROM+AGM with both proofs in prior works. It also serves the educational purpose of helping students understand the Schnorr proof as a stepping stone to the more complicated ECVRF proof. Finally, in Section 4 we present our proof that ECVRF is non-malleable in the ROM+AGM, and compare it with both the Schnorr proof in [5] and the ECVRF proof in [10].

2 Preliminaries

Notation. Let n be the security parameter; we assume that 1^n is given as input to all algorithms and is omitted. For an (efficiently samplable) set S, we write $x \leftarrow S$ for sampling an element x from S uniformly. We write $y \leftarrow \mathcal{A}(x_1, \ldots)$ for the procedure of executing algorithm $\mathcal{A}$ on inputs $(x_1, \ldots)$ and obtaining the output y; if $\mathcal{A}$ is deterministic, we instead write $y := \mathcal{A}(x_1, \ldots)$. We use "PPT" as a shorthand for "probabilistic polynomial-time".

2.1 The Discrete Logarithm Assumption

Let $\mathsf{GenGroup}$ be an algorithm that outputs $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of prime order $p > 2^{n-1}$ for which g is a generator. Let $X \leftarrow \mathbb{G}$ be a random group element. The *discrete logarithm problem* is to compute $\log X = x \in \mathbb{Z}_p$ such that $g^x = X$. Formally, we define the following experiment:

Experiment DL:

1. $(\mathbb{G}, p, g) \leftarrow \mathsf{GenGroup}()$
2. $X \leftarrow \mathbb{G}$
3. Run $x' \leftarrow \mathcal{A}(\mathbb{G}, p, g, X)$
4. $\mathcal{A}$ wins if $x' \in \mathbb{Z}_p$ and $g^{x'} = X$

Fig. 1: Discrete Logarithm Experiment

Definition 1. *We say that the* discrete logarithm (DL) problem is hard *(or the DL assumption holds) relative to* GenGroup *if for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that* $\Pr[\mathcal{A}$ *wins experiment DL*$] \leq \mathsf{negl}(n)$.

Below we abuse notations and say the DL assumption holds for $(\mathbb{G}, p, g)$; furthermore, we assume that $(\mathbb{G}, p, g)$ are given to all parties as part of their inputs and do not explicitly write them. We also assume that all integers and their operations are in $\mathbb{Z}_p$.

2.2 The Algebraic Group Model

The Algebraic Group Model (AGM) is an idealized model that puts some constraints on how an adversary derives any group elements it generates. Specifically, suppose a group $\mathbb{G}$ has been generated and fixed. Whenever the adversary $\mathcal{A}$ outputs a group element $X \in \mathbb{G}$, $\mathcal{A}$ must also return the *algebraic representation* of X, namely a product of powers of group elements that $\mathcal{A}$ has previously seen: if $\mathcal{A}$'s view prior to outputting X consists of $X_1, \ldots, X_\ell \in \mathbb{G}$ then it must return the algebraic representation in the form of

$$X = \prod_{i=1}^{\ell} X_i^{\rho_i}$$

for some $\rho_1, \ldots, \rho_\ell \in \mathbb{Z}_p$.

Since our security proofs are in the combination of ROM and AGM, the adversary must provide the algebraic representations of group elements in its final output *as well as its random oracle queries*. This will be critical in our proof of ECVRF's non-malleability.

2.3 Signature Schemes

We recall the standard definition of signature schemes and their security.

Definition 2. *A* signature scheme *is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *where*

- *The key generation algorithm* Gen *outputs a public/secret key pair* (pk, sk).
- *The signing algorithm* Sign*, on input the secret key* sk *and a message* m*, outputs a signature* σ.
- *The (deterministic) verification algorithm* Ver*, on input the public key* pk*, a message* m *and a signature* σ*, outputs a bit* b *where* 1 *indicates "accept" and* 0 *indicates "reject".*

The correctness property says that for $(pk, sk) \leftarrow \mathsf{Gen}()$, any message m, and $\sigma \leftarrow \mathsf{Sign}(sk, m)$, it holds that $\Pr[\mathsf{Ver}(pk, m, \sigma) = 1] = 1$.

Definition 3. *A signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is* existentially unforgeable under chosen message attacks *(*$\mathsf{EUF\text{-}CMA\text{-}secure}$*, or just* secure*) if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that* $\Pr[\mathcal{A}$ *wins experiment Forge*$] \leq \mathsf{negl}(n)$*. Experiment Forge is defined in Figure 2.*

2.4 The Schnorr Signature Scheme

The Schnorr signature scheme, presented in Figure 3, is a signature scheme based on exponents in prime-order cyclic groups. It is easy to see that the scheme is correct: if $X = g^x$, $R = g^r$, and $s = r + cx$, then $RX^c = g^r(g^x)^c = g^{r+cx} = g^s$.

Experiment Forge:

1. $(pk, sk) \leftarrow \mathsf{Gen}()$. Initialize an empty set $Q := \{\}$ to keep track of $\mathcal{A}$'s oracle queries.
2. $\mathcal{A}(pk)$ is given access to a signing oracle Sign_{sk} which executes the signing algorithm $\mathsf{Sign}(sk, \cdot)$. When $\mathcal{A}$ queries $\mathsf{Sign}_{sk}(m)$, update $Q := Q \cup \{m\}$.
3. $\mathcal{A}$ outputs a message m^* and a signature σ^*.
4. $\mathcal{A}$ wins if and only if $\mathsf{Ver}(pk, m^*, \sigma^*) = 1$ and $m^* \notin Q$ (i.e., the message was not previously queried).

Fig. 2: The Signature Security Experiment

Public parameters: cyclic group $\mathbb{G}$ with generator g and prime order p, and $H : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$ is a hash function modeled as a random oracle.

$\mathsf{Gen}()$:

1. $x \leftarrow \mathbb{Z}_p$; $X := g^x$
2. Output $(pk := X, sk := x)$

$\mathsf{Sign}(x, m)$:

1. $r \leftarrow \mathbb{Z}_p$; $R := g^r$
2. $c := H(R, m)$
3. $s := r + cx$
4. Output $\sigma := (R, s)$

$\mathsf{Ver}(X, m, (R, s))$:

1. $c := H(R, m)$
2. If $RX^c = g^s$ output 1; else output 0

Fig. 3: The Schnorr Signature Scheme

2.5 VRF and Non-Malleability

Conceptually, a *verifiable random function (VRF)* can be viewed as an asymmetric version of a pseudorandom function: a private key can be used to generate pseudorandom outputs from a message, and the public key can be used to verify that the output was generated correctly.

While a VRF has multiple security properties, in this paper we only consider *non-malleability*. In this context, a VRF can be defined just as a signature scheme (Definition 2); the only difference is that the signing algorithm Sign of a signature scheme is called the proving algorithm Prove here, and the signature σ is called the proof π.

Non-malleability is almost identical to the security of a signature scheme, with just one difference: under EUF-CMA security the adversary may not output a previously-queried message but may output a previously-seen signature; whereas under non-malleability the adversary may output a previously-queried message but may not reuse a previously-seen proof.

Definition 4. *A VRF* $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ *is* $\mathsf{non\text{-}malleable}$ *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that* $\Pr[\mathcal{A}$ *wins experiment NM*$] \leq \mathsf{negl}(n)$*. Experiment NM is defined in Figure 4.*

Experiment NM:

1. $(pk, sk) \leftarrow \mathsf{Gen}()$. Initialize an empty set $P := \{\}$ to keep track of the results of $\mathcal{A}$'s oracle query outputs.
2. $\mathcal{A}(pk)$ is given access to a proof oracle Prove_{sk} which executes the proving algorithm $\mathsf{Prove}(sk, \cdot)$. When $\mathcal{A}$ queries $\mathsf{Prove}_{sk}(m)$ whose output is π, update $P := P \cup \{\pi\}$.
3. $\mathcal{A}$ outputs a message m^* and a proof π^*.
4. $\mathcal{A}$ wins iff $\mathsf{Ver}(pk, m^*, \pi^*) = 1$ and $\pi^* \notin P$ (i.e., $\mathcal{A}$ does not repeat a previously-seen proof).

Fig. 4: The VRF Non-Malleability Experiment

2.6 ECVRF

ECVRF (Elliptic Curve VRF) is, as the name suggests, a VRF implemented on an elliptic curve, and similarly to the Schnorr signature scheme, is based on exponents in cyclic groups. It is presented in Figure 5.

It is easy to see that the scheme is correct: if $X = g^x$, $Z = Y^x$, and $s = r + cx$, then in verification $R = g^s / X^c = g^s / (g^x)^c = g^{s-cx} = g^r$ and $R_Y = Y^s / Z^c =$

$Y^s/(Y^x)^c = Y^{s-cx} = Y^r$, so $H(Y, Z, R, R_Y) = H(Y, Z, g^r, Y^r) = c$ (the second equation is because c is defined as $H(Y, Z, g^r, Y^r)$ in the proving algorithm).

Public parameters: cyclic group $\mathbb{G}$ with generator g and prime order p; HTC (for "hash to curve") : $\mathbb{G} \times \{0,1\}^* \to \mathbb{G}$ and $H : \mathbb{G}^4 \to \mathbf{H}$ (where $\mathbf{H} \subset \mathbb{G}$ and $|\mathbf{H}|$ is super-polynomial) are two hash functions modeled as random oracles.

Gen():

1. $x \leftarrow \mathbb{Z}_p$; $X := g^x$
2. Output $(pk := X, sk := x)$

Prove(x, m):

1. $Y := HTC(g^x, m)$; $Z := Y^x$
2. $r \leftarrow \mathbb{Z}_p$
3. $R := g^r$; $R_Y := Y^r$
4. $c := H(Y, Z, R, R_Y)$
5. $s := r + cx$
6. Output (Z, c, s)

Ver$(X, m, (Z, c, s))$:

1. $Y := HTC(X, m)$
2. $R := g^s/X^c$; $R_Y := Y^s/Z^c$
3. If $H(Y, Z, R, R_Y) = c$ output 1; else output 0

Fig. 5: ECVRF

Comparison with the Schnorr signature scheme. ECVRF uses a similar idea to the Schnorr signature scheme, and we briefly mention their differences below:

– In ECVRF, aside from the public group generator g, we use a hash-to-curve function on the message to select another generator Y, and construct a public key analog $Z = Y^x$ (this value is returned as part of the proof). Furthermore, analogous to $R = g^r$ in the Schnorr signature scheme, we

additionally compute group element $R_Y = Y^r$.
Finally, the inputs to the random oracle H contain the additional group element Z and R_Y, and the message m (which is an H input in the Schnorr scheme) is replaced by its "representation in the group" Y.
- In the Schnorr scheme, the signature contains R and s, and the H output c is the intermediate value; in verification we recompute c and check if R satisfies the equation. In ECVRF, the proof contains c and s, which we use in conjunction with other known values to recompute R and R_Y; after that we check if the H function returns the same value c — an ordering opposite that of Schnorr.
- In the Schnorr scheme, the range of H (or equivalently, the range of c) is $\mathbb{Z}_p$, whereas in ECVRF it is a large subset of $\mathbb{Z}_p$, $\mathbf{H}$.

Finally, we note that in an implementation of ECVRF, the verification algorithm and the random oracles HTC, H might take as input elements of a larger group $\mathbb{E}$ rather than $\mathbb{G}$, where $\mathbb{G}$ is a subgroup of $\mathbb{E}$ and the co-factor $|\mathbb{E}|/|\mathbb{G}|$ is co-prime with $|\mathbb{G}|$. (The key generation and proving algorithms are unchanged, and the range of HTC is still $\mathbb{G}$.) This allows for more efficient verification if checking group membership is slow in $\mathbb{G}$ but fast in $\mathbb{E}$.[1] We present our non-malleability proof in the setting that all group elements are in $\mathbb{G}$, but explain after the proof why it still holds even if inputs to verification can be elements of the larger group $\mathbb{E}$.

3 Schnorr Signature Scheme in the ROM+AGM

In this section, we rewrite the Schnorr signature security proof in [5].

Theorem 1. *If the DL assumption holds for $(\mathbb{G}, p, g)$, then the Schnorr signature scheme (Figure 3) is secure in the ROM+AGM.*

The proof of the theorem is done via a reduction to the DL assumption. The reduction works in two steps:

- The reduction first needs to simulate the forgery experiment to the adversary. The reduction embeds its DL challenge X as the public key; it can simulate the signing oracle without knowledge of the secret key via choosing random s and c, setting $R := g^s/X^c$, and then "programming" the random oracle H by setting $H(R, m) := c$. Note that this step does not require the adversary to be algebraic.
- After the adversary outputs the forgery $(m^*, (R^*, s^*))$, the reduction computes $x = \log X$ via two expressions of R^* using g and X, one from the adversary's winning condition, i.e.,

$$R^* = \frac{g^{s^*}}{X^{c^*}}$$

[1] We stress that group membership checks, while not a focus of our work, must be performed in practice.

(where $c^* = H(R^*, m^*)$); and the other from the algebraic representation of R^* given by the adversary. The set of group elements seen by the adversary is: the generator g, the public key X, and the R_i ($i \in [q]$ where q is the number of the adversary's signing oracle queries) values returned by the signing oracle; so the adversary's algebraic representation of R^* must be expressed as a product of powers of these group elements, i.e.,

$$R^* = g^{\gamma} X^{\xi} \prod_{i=1}^{q} R_i^{\rho_i}.$$

From these two expressions, we can solve

$$x = \frac{s^* - \gamma - \sum_{i=1}^{q} \rho_i s_i}{c^* + \xi - \sum_{i=1}^{q} \rho_i c_i}.$$

The reduction would fail if $c^* + \xi - \sum_{i=1}^{q} \rho_i c_i = 0$. However, the probability of this occurring is negligible. This is because $c^* = H(R^*, m^*)$ and $c_i = H(R_i, m_i)$, and the adversary's winning condition implies $m^* \neq m_i$, so c^* is independent of c_i; furthermore, the adversary must have chosen R^* — and thus must have chosen $\xi, \rho_1, \ldots, \rho_q$ — *before* querying H to obtain c^*, so c^* is also independent of $\xi, \rho_1, \ldots, \rho_q$. In sum, c^* is a random integer that is independent of all other terms in the denominator, so the probability that the denominator is 0 is negligible.

Proof. Fix an arbitrary PPT adversary $\mathcal{A}$ in experiment Forge for the Schnorr signature scheme; assume $\mathcal{A}$ makes q queries to the signing oracle Sign_x and q_H queries to the random oracle H. We construct a reduction $\mathcal{B}$ in Figure 8 that uses $\mathcal{A}$ to solve the DL problem for $(\mathbb{G}, p, g)$.

Clearly $\mathcal{B}$ is PPT. We now analyze the probability that $\mathcal{B}$'s output x' is equal to x.

Simulating experiment Forge.

Claim. If $\mathsf{TwoROValues}$ does not occur, then $\mathcal{A}$'s view simulated by $\mathcal{B}$ is identical to $\mathcal{A}$'s "real" view in experiment Forge.

Proof. $\mathcal{A}$'s view consists of three parts: the public key X, answers to the signing oracle Sign_x, and answers to the random oracle H. In both views, X is a random element of $\mathbb{G}$, and H queries are answered via lazy sampling (except for the "programming" while answering Sign_x queries). As for answers to Sign_x, we first define an intermediate signing oracle in Figure 9.

$\mathsf{Sign}_x(m_i)$:

1. $c_i, s_i \leftarrow \mathbb{Z}_p$
2. $R_i := g^{s_i}/X^{c_i}$
3. If $H(R_i, m_i)$ is defined, output $\mathsf{TwoROValues}$ and abort
4. Program $\underline{H(R_i, m_i) := c_i}$
5. Return (R_i, s_i) to $\mathcal{A}$

Fig. 6: Signing Oracle

$H(R, m)$:

1. If $H(R, m)$ is already defined, return that value to $\mathcal{A}$
2. $\underline{H(R, m) \leftarrow \mathbb{Z}_p}$; return $H(R, m)$ to $\mathcal{A}$

Fig. 7: Random Oracle

$\mathcal{B}(X)$:

1. Run $\mathcal{A}(X)$, simulating $\mathcal{A}$'s oracle queries as described in Figures 6 and 7
2. $\mathcal{A}$ outputs $(m^*, (R^*, s^*))$ together with the algebraic representation of R^*, i.e., $R^* = g^{\gamma} X^{\xi} \prod_{i=1}^{q} R_i^{\rho_i}$
3. If $m^* = m_i$ for some $i \in [q]$, abort
4. If $H(R^*, m^*)$ is undefined, choose $\underline{H(R^*, m^*) \leftarrow \mathbb{Z}_p}$
5. $c^* := H(R^*, m^*)$
6. If $c^* + \xi - \prod_{i=1}^{q} \rho_i c_i = 0$, output $\mathsf{ZeroDenominator}$ and abort
7. Output
$$x' := \frac{s^* - \gamma - \sum_{i=1}^{q} \rho_i s_i}{c^* + \xi - \sum_{i=1}^{q} \rho_i c_i}$$

Fig. 8: Reduction $\mathcal{B}$ that solves the DL problem (underlined texts show situations where an H output is defined)

$\mathsf{Sign}_x(m_i)$:

1. $c_i \leftarrow \mathbb{Z}_p$
2. $r_i \leftarrow \mathbb{Z}_p$; $R_i := g^{r_i}$
3. If $H(R_i, m_i)$ is defined, output TwoROValues and abort
4. Program $H(R_i, m_i) := c_i$
5. $s_i := r_i + c_i x$
6. Return (R_i, s_i) to $\mathcal{A}$

Fig. 9: Intermediate Signing Oracle

The difference between this and the "real" signing oracle in experiment Forge is the addition of line 3 and that c_i is randomly sampled in advance and then used to "program" $H(R_i, m_i)$ (whereas in the "real" signing oracle $c_i := H(R_i, m_i)$ is chosen when H is queried). Since TwoROValues does not occur by assumption, we can ignore line 3. Furthermore, since TwoROValues does not occur, $H(R_i, m_i)$ has not yet been sampled, so the intermediate approach of "pre-programming" H is no different than simply querying H. Therefore the two signing oracles are identical to $\mathcal{A}$.

Next, in both the intermediate signing oracle and the signing oracle simulated by $\mathcal{B}$ (Figure 6), c_i is a random integer in $\mathbb{Z}_p$ and R_i, s_i satisfy $g^{s_i} = R_i X^{c_i}$. If we choose $R_i \leftarrow \mathbb{G}$ then we get the intermediate signing oracle, and if we choose $s_i \leftarrow \mathbb{Z}_p$ then we get the signing oracle simulated by $\mathcal{B}$. Since $|\mathbb{G}| = |\mathbb{Z}_p| = p$, these two approaches are identical in the view of $\mathcal{A}$. We conclude that $\mathcal{A}$'s view simulated by $\mathcal{B}$ is identical to $\mathcal{A}$'s "real" view in experiment Forge.

Claim. $\Pr[\mathsf{TwoROValues}] \leq \frac{q(q+2q_H-1)}{2p}$.

Proof. TwoROValues occurs if there exists a query (say the i-th query) m_i to Sign_x by $\mathcal{A}$ where $H(R_i, m_i)$ is already defined. Recall that each query to H explicitly defines an H value and each query to Sign_x implicitly defines an H value (see underlined texts in Figure 8); thus, when the i-th query to Sign_x is made, we have defined at most $q_H + i - 1$ H values. Since R_i is a random element of $\mathbb{G}$ (independent of everything else), the probability that it is equal to one of the $q_H + i - 1$ R values that appear as an H input, is at most $(q_H + i - 1)/p$. Applying a union bound, we get

$$\Pr[\mathsf{TwoROValues}] \leq \sum_{i=1}^{q} \frac{q_H + i - 1}{p} = \frac{q(q_H - 1)}{p} + \frac{q(q+1)}{2p} = \frac{q(q + 2q_H - 1)}{2p}.$$

Computing x after $\mathcal{A}$ halts.

Claim. If ZeroDenominator does not occur, and $\mathcal{A}$ wins experiment Forge, then $\mathcal{B}$ outputs $x' = x$.

Proof. If successful, $\mathcal{A}$ returns $(m^*, (R^*, s^*))$ where $R^* = g^{s^*}/X^{c^*}$ and $c^* = H(R^*, m^*)$, together with an algebraic representation $R^* = g^{\gamma} X^{\xi} \prod_{i=1}^{q} R_i^{\rho_i}$. From these two expressions, we get

$$\frac{g^{s^*}}{X^{c^*}} = g^{\gamma} X^{\xi} \prod_{i=1}^{q} R_i^{\rho_i} = g^{\gamma} X^{\xi} \prod_{i=1}^{q} \left(\frac{g^{s_i}}{X^{c_i}} \right)^{\rho_i} = g^{\gamma + \sum_{i=1}^{q} \rho_i s_i} X^{\xi - \sum_{i=1}^{q} \rho_i c_i}$$

(where the second equation is due to how R_i is defined), or equivalently,

$$g^{s^* - \gamma - \sum_{i=1}^{q} \rho_i s_i} = X^{c^* + \xi - \sum_{i=1}^{q} \rho_i c_i},$$

from which we can solve for x as

$$x = \frac{s^* - \gamma - \sum_{i=1}^{q} \rho_i s_i}{c^* + \xi - \sum_{i=1}^{q} \rho_i c_i}$$

as $\mathcal{B}$ does. Note that this expression is well-defined as long as $c^* + \xi - \sum_{i=1}^{q} \rho_i c_i \neq 0$ (i.e., $\mathsf{ZeroDenominator}$ does not occur), since p is prime.

Claim. $\Pr[\mathsf{ZeroDenominator}] \leq \frac{q_H + 1}{p}$.

Proof. $\mathsf{ZeroDenominator}$ occurs if and only if

$$c^* + \xi - \sum_{i=1}^{q} \rho_i c_i = 0 \tag{1}$$

when $\mathcal{B}$ attempts to compute x after $\mathcal{A}$ halts. Recall that $c^* = H(R^*, m^*)$ takes on a value from one of three places where an H output is defined (see underlined texts in Figure 8):

1. A query to Sign_x by $\mathcal{A}$;
2. An explicit query to H by $\mathcal{A}$; and
3. At the end of the experiment after $\mathcal{A}$ halts.

First, we show that case 1 is impossible. Suppose that c^* is defined in the i-th query to the signing oracle by $\mathcal{A}$. That is, $c^* = H(R^*, m^*)$ is defined as $H(R_i, m_i)$, which implies that $m^* = m_i$ — but then $\mathcal{B}$ would have already aborted (line 3 of Figure 8). So this case is ruled out.

Thus, c^* must be defined in either case 2 or case 3, so it takes on one of (at most) $q_H + 1$ values (q_H values in case 2 and 1 value in case 3). Note that c^* and $c_1, \ldots, c_q$ are independently sampled while simulating H, since $c^* = H(R^*, m^*)$, $c_i = H(R_i, m_i)$ and $m^* \neq m_i$ for all $i \in [q]$. Also, the algebraic representation of R^* — which includes ξ and $\rho_1, \ldots, \rho_q$ — is determined either when $\mathcal{A}$ queries H on R^* (case 2), or when $\mathcal{A}$ outputs R^* as part of its forgery (case 3). Regardless, this happens before c^* is chosen at random. Therefore c^* is independent of ξ and $\rho_1, \ldots, \rho_q$. We conclude that c^* is a random integer in $\mathbb{Z}_p$ independent of all other variables in (1). So for each of the $q_H + 1$ possible values for c^*, (1) holds

with probability $1/p$. By the union bound, (1) occurs with probability $(q_H+1)/p$ and the result follows.[2]

In sum, we have proved that $\mathcal{B}$ wins experiment DL as long as $\mathcal{A}$ wins experiment Forge and the two "bad events" TwoROValues and ZeroDenominator do not occur. Therefore,

$$\begin{aligned}\Pr[\mathcal{B}\text{ wins}] &\geq \Pr[\mathcal{A}\text{ wins}] - \Pr[\mathsf{ZeroDenominator}] - \Pr[\mathsf{TwoROValues}] \\ &\geq \Pr[\mathcal{A}\text{ wins}] - \frac{q(q+2q_H-1)+2q_H+2}{2p}.\end{aligned}$$

Since $\Pr[\mathcal{B}\text{ wins}]$ is negligible as the DL assumption holds for $(\mathbb{G}, p, g)$, we know that $\Pr[\mathcal{A}\text{ wins}]$ is also negligible. This completes the proof. ■

Comparison with the proof in [5]. Our proof is fundamentally the same as that in [5]. Both proofs essentially use the same reduction to reduce Schorr signature security to the discrete logarithm assumption, and the reduction fails under the same "bad events": the denominator is 0 in the expression for x (ZeroDenominator), or the simulated signing oracle cannot set a desired output of the random oracle H because the adversary has already queried H on the same input (TwoROValues). However, our proof differs from theirs in expression. The proof in [5] uses the "game-hop" methodology: it begins with Game_0 which is the adversary's real view, rules out the "bad event" ZeroDenominator in Game_1 and TwoROValues in Game_2, and finally reduces to DL in Game_2; in other words, it rules out the "bad events" *before* presenting the reduction, while we let the reduction abort if either of the "bad events" occurs. The proof in [5] is more concise; in particular, we provide a detailed argument why the reduction's simulation of the signing oracle is identical to the adversary's real view if TwoROValues does not occur. Furthermore, our upper bound of $\Pr[\mathsf{TwoROValues}]$ is slightly tighter than theirs ($(q(q+2q_H-1))/(2p)$ versus $q(q+q_H)/p$). Note also that we use multiplicative notation to denote group operations, whereas [5] uses additive notation.

4 Non-Malleability of ECVRF in the ROM+AGM

In this section, we present a *tight* non-malleability proof for ECVRF in the ROM+AGM.

Theorem 2. *If the DL assumption holds for $(\mathbb{G}, p, g)$, then ECVRF (Figure 5) is non-malleable in the ROM+AGM.*

[2] Note that the upper bound of $\Pr[\mathsf{ZeroDenominator}]$ is not $1/p$, since $\mathcal{A}$ can make q_H H queries and obtain q_H candidate c^* values, and if *any* of them makes (1) happen, $\mathcal{A}$ can choose that value as c^*.

The proof of this theorem is similar to that of Schnorr signature security, and we only highlight the differences here. It is done via a reduction to the DL assumption, and just as in the Schnorr proof, the reduction works in two steps:

- The reduction first needs to simulate the non-malleability experiment to the adversary. As in the Schnorr security proof, the reduction simulates the proof oracle without knowledge of the secret key by choosing random c, s and "programming" the random oracle H accordingly. In addition, while simulating the proof oracle, the reduction needs to come up with random group elements Y, Z where $Y = HTC(X, m)$ and $Z = Y^x$. In order to achieve this without knowing x, the reduction chooses a random integer y as the "discrete logarithm trapdoor", and sets $Y = g^y$ and $Z = X^y$.
- After the adversary outputs the message/proof pair $(m^*, (Z^*, c^*, s^*))$, the reduction can first compute the intermediate values Y^*, R^*, and R^*_Y as in verification. In order for the adversary to win with non-negligible probability, $H(Y^*, Z^*, R^*, R^*_Y)$ must have been defined to be c^* before the adversary halts (otherwise the result of $H(Y^*, Z^*, R^*, R^*_Y)$ is chosen at random *after* the adversary outputs c^*, so the adversary has a negligible probability of "predicting" c^* that is equal to $H(Y^*, Z^*, R^*, R^*_Y)$). Furthermore, it is not hard to see that if c^* is implicitly defined as one of the c_i values returned by the proof oracle, then the adversary has repeated the proof and thus lost, so this case can also be ruled out. The remaining case is that c^* is result of an explicit $H(Y^*, Z^*, R^*, R^*_Y)$ query by the adversary; call it the "crucial query". Then the reduction can compute $x = \log X$ via two expressions of R^* using g and X, one from the definition of R^*, i.e.,

$$R^* = \frac{g^{s^*}}{X^{c^*}},$$

and the other from the algebraic representation of R^* given by the adversary (note that the adversary must provide this algebraic representation while making the crucial query, since R^* is part of its random oracle query inputs). The set of group elements seen by the adversary is: the generator g, the public key X, the Z_i ($i \in [q']$ where q' is the number of the adversary's proof oracle queries before its crucial query) values returned by the proof oracle, and the Y'_j ($j \in [q'_{HTC}]$ where q'_{HTC} is the number of the adversary's HTC queries before its crucial query) values from the adversary's HTC queries; so the adversary's algebraic representation of R^* must be expressed as a product of powers of these group elements, i.e.,

$$R^* = g^\gamma X^\xi \prod_{j=1}^{q'_{HTC}} (Y'_j)^{\alpha_j} \prod_{i=1}^{q'} Z_i^{\beta_i}.$$

The reduction knows both $y'_j = \log_g Y'_j$ (the reduction can choose it while simulating the adversary's HTC queries) and $y_i = \log_X Z_i$ (which is the "discrete logarithm trapdoor" while simulating the adversary's proof oracle

queries — see the bullet above), so it can rewrite this expression using g and X only. From these two expressions, we can solve

$$x = \frac{s^* - \gamma - \sum_{j=1}^{q'_{HTC}} \alpha_j y'_j}{c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i}.$$

The reduction would fail if $c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i = 0$. However, the probability of this occurring is negligible because c^* is independent of all the other items in the denominator: similar to the proof for the Schnorr signature scheme, c^* is independent of the algebraic representations $\xi, \beta_1, \ldots, \beta_{q'}$ since they are determined by the adversary *before* making the crucial query. As for y_i, first note that c^* is generated after y_i are defined (since we have ruled out the possibility that c^* is defined implicitly during a proof oracle query); at the time when c^* is generated, the H outputs that have been defined include (1) outputs of the adversary's explicit H queries and (2) those implicitly defined during the adversary's proof oracle queries, all of which have inputs different from the crucial query (otherwise c^* would have been defined previously). Therefore, the process of generating c^* is independent of what appeared in the experiment previously, including y_i.

Proof. Fix an arbitrary PPT adversary $\mathcal{A}$ in the non-malleability experiment for ECVRF; assume $\mathcal{A}$ makes q queries to the proof oracle Prove_x, q_{HTC} queries to the random oracle HTC, and q_H queries to the random oracle H. We construct a reduction $\mathcal{B}$ in Figures 13 and 14 that uses $\mathcal{A}$ to solve the DL problem for $(\mathbb{G}, p, g)$.

Clearly $\mathcal{B}$ is PPT. We now analyze the probability that $\mathcal{B}$'s output x' is equal to x.

Simulating experiment NM.

Claim. If $\mathsf{TwoROValues}$ does not occur, then $\mathcal{A}$'s view simulated by $\mathcal{B}$ is identical to $\mathcal{A}$'s "real" view in experiment NM.

Proof. $\mathcal{A}$'s view consists of four parts: the public key X, answers to the proof oracle Prove_x, and answers to the random oracles HTC and H. In both views, X is a random element of $\mathbb{G}$, and HTC and H queries are answered via lazy sampling (except for the "programming" of H while answering Prove_x queries). As for answers to Prove_x, we first define an intermediate proof oracle in Figure 15.

The difference between this and the "real" proof oracle in experiment NM is the addition of line 4 and that c_i is randomly sampled in advance and then used to "program" $H(Y_i, Z_i, R_i, R_{Y,i})$ (whereas in experiment NM $c_i := H(Y_i, Z_i, R_i, R_{Y,i})$ is chosen when H is queried). Similar to the proof of the corresponding claim for the Schnorr signature scheme, the two proof oracles are identical to $\mathcal{A}$.

We now define a second intermediate proof oracle in Figure 16.

$\mathsf{Prove}_x(m_i)$:

1. If $HTC(X, m_i)$ is defined, retrieve record $\langle m_i, y_i \rangle$
2. Else $y_i \leftarrow \mathbb{Z}_p$ and program $HTC(X, m_i) := g^{y_i}$
3. $Y_i := g^{y_i}$; $Z_i := X^{y_i}$
4. $c_i \leftarrow \mathbf{H}$; $s_i \leftarrow \mathbb{Z}_p$
5. $R_i := g^{s_i} / X^{c_i}$
6. $R_{Y,i} := R_i^{y_i}$
7. If $H(Y_i, Z_i, R_i, R_{Y,i})$ is defined, output $\mathsf{TwoROValues}$ and abort
8. Program $\underline{H(Y_i, Z_i, R_i, R_{Y,i}) := c_i}$
9. Return $(\underline{Z_i}, c_i, s_i)$ to $\mathcal{A}$

Fig. 10: Proof Oracle

$HTC(X'_j, m'_j)$:

1. If $HTC(X'_j, m'_j)$ is already defined, return that value to $\mathcal{A}$
2. Choose $y'_j \leftarrow \mathbb{Z}_p$, define $Y'_j := HTC(X'_j, m'_j) := g^{y'_j}$, and return Y'_j to $\mathcal{A}$. Furthermore, if $X'_j = X$ then record $\langle m'_j, y'_j \rangle$

Fig. 11: Random Oracle HTC

$H(Y, Z, R, R_Y)$:

1. If $H(Y, Z, R, R_Y)$ is already defined, return that value to $\mathcal{A}$
2. $\underline{H(Y, Z, R, R_Y) \leftarrow \mathbb{Z}_p}$; return $H(Y, Z, R, R_Y)$ to $\mathcal{A}$

Fig. 12: Random Oracle H

Fig. 13: Reduction $\mathcal{B}$ simulating $\mathcal{A}$'s oracle queries (underlined texts show situations where an H output is defined)

$\mathcal{B}(X)$:

1. Run $\mathcal{A}(X)$, simulating $\mathcal{A}$'s oracle queries as described in Figures 10 to 12
2. $\mathcal{A}$ outputs $(m^*, (Z^*, c^*, s^*))$
3. If $(Z^*, c^*, s^*) = (Z_i, c_i, s_i)$ for some $i \in [q]$, abort
4. $Y^* := HTC(X, m^*)$ (choose $Y^* \leftarrow \mathbb{G}$ if $HTC(X, m^*)$ is undefined); $R^* := g^{s^*}/X^{c^*}$; $R_Y^* := (Y^*)^{s^*}/(Z^*)^{c^*}$
5. Retrieve the $H(Y^*, Z^*, R^*, R_Y^*)$ query by $\mathcal{A}$. If there is no such query, or if the output of the query is not c^*, output NoROQuery and abort. Call this H query the "crucial query"
6. Retrieve the algebraic representation of R^*, i.e, $R^* = g^{\gamma} X^{\xi} \prod_{j=1}^{q'_{HTC}} (Y'_j)^{\alpha_j} \prod_{i=1}^{q'} Z_i^{\beta_i}$ (where q' and q'_{HTC} are the numbers of Prove and HTC queries respectively before the crucial query)
7. If $c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i = 0$, output ZeroDenominator and abort
8. Output

$$x' = \frac{s^* - \gamma - \sum_{j=1}^{q'_{HTC}} \alpha_j y'_j}{c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i}$$

Fig. 14: Reduction $\mathcal{B}$ solving the DL problem after $\mathcal{A}$ halts

$\mathsf{Prove}_x(m_i)$:

1. $Y_i := HTC(X, m_i)$; $Z_i := Y_i^x$
2. $c_i \leftarrow \mathbf{H}$
3. $r_i \leftarrow \mathbb{Z}_p$; $R_i := g^{r_i}$; $R_{Y,i} := Y_i^{r_i}$
4. If $H(Y_i, Z_i, R_i, R_{Y,i})$ is defined, output TwoROValues and abort
5. Program $H(Y_i, Z_i, R_i, R_{Y,i}) := c_i$
6. $s_i := r_i + c_i x$
7. Return (Z_i, c_i, s_i) to $\mathcal{A}$

Fig. 15: Intermediate proof oracle I

$\mathsf{Prove}_x(m_i)$:

1. $Y_i := HTC(X, m_i)$; $Z_i := Y_i^x$
2. $c_i \leftarrow \mathbf{H}$; $s_i \leftarrow \mathbb{Z}_p$
3. $r_i := s_i - c_i x$; $R_i := g^{s_i}/X^{c_i}$; $R_{Y,i} := Y_i^{r_i}$
4. If $H(Y_i, Z_i, R_i, R_{Y,i})$ is defined, output $\mathsf{TwoROValues}$ and abort
5. Program $H(Y_i, Z_i, R_i, R_{Y,i}) := c_i$
6. $s_i := r_i + c_i x$
7. Return (Z_i, c_i, s_i) to $\mathcal{A}$

Fig. 16: Intermediate proof oracle II

In both intermediate proof oracles c_i is a random integer in $\mathbf{H}$, R_i, s_i satisfy $g^{s_i} = R_i X^{c_i}$, and $r_i = \log R_i$. If we choose $R_i \leftarrow \mathbb{G}$ then we get the first intermediate proof oracle, and if we choose $s_i \leftarrow \mathbb{Z}_p$ then we get the second intermediate proof oracle. Since $|\mathbb{G}| = |\mathbb{Z}_p| = p$, these two approaches are identical in the view of $\mathcal{A}$.[3]

The difference between the second intermediate proof oracle and the proof oracle simulated by $\mathcal{B}$ (Figure 10) is that the former defines $Z_i = Y_i^x$ and $R_{Y,i} = Y_i^{r_i}$, whereas the latter defines $Z_i = X^{y_i}$ and $R_{Y,i} = R_i^{y_i}$. However, in both cases $Z_i = g^{x \log Y_i}$ and $R_{Y,i} = g^{\log R_i \log Y_i}$ (note that $Y_i = HTC(X, m_i)$ and $R_i = g^{s_i}/X^{c_i}$ are defined in the same manner in the two cases), so there is no difference. We conclude that $\mathcal{A}$'s view simulated by $\mathcal{B}$ is identical to $\mathcal{A}$'s "real" view in experiment NM.

Claim. $\Pr[\mathsf{TwoROValues}] \leq \frac{q(q+2q_H-1)}{2p}$.

Proof. The proof is very similar to that of the corresponding claim for the Schnorr signature scheme, so we only provide a sketch here. $\mathsf{TwoROValues}$ occurs if there exists a query (say the i-th query) m_i to Prove_x by $\mathcal{A}$ in which $H(Y_i, Z_i, R_i, R_{Y,i})$ is already defined. At this moment we have defined at most $q_H + i - 1$ H values, so the probability that R_i is one of them is at most $(q_H + i - 1)/p$ (note that R_i is a random element of $\mathbb{G}$ even though c_i is chosen from $\mathbf{H}$ rather than $\mathbb{Z}_p$). Applying a union bound yields the result.

Computing x after $\mathcal{A}$ halts.

Claim. If $\mathsf{ZeroDenominator}$ and $\mathsf{NoROQuery}$ do not occur, and $\mathcal{A}$ wins experiment NM, then $\mathcal{B}$ outputs $x' = x$.

[3] Note that here the range of c_i is $\mathbf{H}$, rather than $\mathbb{Z}_p$ as in the Schnorr signature scheme. However, the same argument still holds; the key point is that s_i has the "full range" $\mathbb{Z}_p$.

Proof. If successful, $\mathcal{A}$ outputs (Z^*, c^*, s^*), and, by our assumption, there exists an H query (the crucial query) such that $H(Y^*, Z^*, R^*, R^*_Y) = c^*$, where R^* is defined as

$$R^* = \frac{g^{s^*}}{X^{c^*}}.$$

Since $\mathcal{A}$ is algebraic, it must provide algebraic representations of all these input values, including $R^* = g^{\gamma} X^{\xi} \prod_{j=1}^{q'_{HTC}} (Y'_j)^{\alpha_j} \prod_{i=1}^{q'} Z_i^{\beta_i}$ (line 6 of Figure 14). Note that $\mathcal{B}$ defines $Y'_j = g^{y'_j}$ (line 2 of Figure 11) and $Z_i = X^{y_i}$ (line 3 of Figure 10). So R^* can be expressed as

$$R^* = g^{\gamma} X^{\xi} \prod_{j=1}^{q'_{HTC}} (g^{y'_j})^{\alpha_j} \prod_{i=1}^{q'} (X^{y_i})^{\beta_i} = g^{\gamma + \sum_{j=1}^{q'_{HTC}} \alpha_j y'_j} X^{\xi + \sum_{i=1}^{q'} \beta_i y_i}.$$

Combining the two expressions above, we get

$$\frac{g^{s^*}}{X^{c^*}} = g^{\gamma + \sum_{j=1}^{q'_{HTC}} \alpha_j y'_j} X^{\xi + \sum_{i=1}^{q'} \beta_i y_i},$$

or equivalently,

$$g^{s^* - \gamma - \sum_{j=1}^{q'_{HTC}} \alpha_j y'_j} = X^{c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i},$$

from which we can solve for x as

$$x = \frac{s^* - \gamma - \sum_{j=1}^{q'_{HTC}} \alpha_j y'_j}{c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i}$$

as $\mathcal{B}$ does. Note that this expression is well-defined as long as $c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i \neq 0$ (i.e., $\mathsf{ZeroDenominator}$ does not occur), since p is prime.

Claim. $\Pr[\mathsf{ZeroDenominator}] \leq \frac{q_H}{|\mathbf{H}|}$.

Proof. $\mathsf{ZeroDenominator}$ occurs if and only if

$$c^* + \xi + \sum_{i=1}^{q'} \beta_i y_i = 0 \tag{2}$$

when $\mathcal{B}$ attempts to compute x after $\mathcal{A}$ halts. Recall that $c^* = H(Y^*, Z^*, R^*, R^*_Y)$ is the output of $\mathcal{A}$'s crucial query (line 5 of Figure 14), and an H output is defined in either of the following two places (see underlined texts in Figure 13):

1. A query to Prove_x by $\mathcal{A}$; and
2. An explicit query to H by $\mathcal{A}$.

We first show that c^* cannot be defined in case 1, i.e., $\mathcal{A}$'s crucial query cannot be identical to one of the previously defined H values while simulating Prove_x. If $c^* = H(Y^*, Z^*, R^*, R^*_Y)$ is defined as $H(Y_i, Z_i, R_i, R_{Y,i})$ for some

$i \in [q]$, then $(Y^*, Z^*, R^*, R^*_Y) = (Y_i, Z_i, R_i, R_{Y,i})$. This implies that $Z^* = Z_i$. Furthermore, since $R^* = R_i$, we have that $g^{s^*}/X^{c^*} = g^{s_i}/X^{c_i}$, and as $c^* = H(Y^*, Z^*, R^*, R^*_Y) = H(Y_i, Z_i, R_i, R_{Y,i}) = c_i$ we have $s^* = s_i$. That means $(Z^*, c^*, s^*) = (Z_i, c_i, s_i)$ which is a contradiction since $\mathcal{B}$ would have already aborted (in line 3 of Figure 14). So this case is ruled out.

Thus, c^* must be defined in case 2, so it takes on one of (at most) q_H values. Consider the moment when c^* is defined; we have just argued that c^* is defined via an explicit query to H (the crucial query) by $\mathcal{A}$. Since the inputs of the crucial query are different from all previous H inputs that resulted in an existing H output (otherwise c^* would have been defined implicitly, i.e., in case 1), we know that c^* is independent of all procedures that occurred previously in the experiment, including y_i for all $i \in [q]$. Moreover, the algebraic representation of R^* — which includes ξ and $\beta_1, \ldots, \beta_{q'}$ — is determined when $\mathcal{A}$ makes its crucial query on R^*, and that happens before c^* is defined. Therefore c^* is independent of ξ and $\beta_1, \ldots, \beta_{q'}$. We conclude that c^* is a random integer in $\mathbf{H}$ independent of all other variables in (2). So for each of the q_H possible values for c^*, (2) holds with probability $1/|\mathbf{H}|$. By the union bound, (2) occurs with probability $q_H/|\mathbf{H}|$ and the result follows.

Claim. $\Pr[\mathsf{NoROQuery} \wedge \mathcal{A} \text{ wins}] \leq \frac{1}{|\mathbf{H}|}$.

Proof. Assume that NoROQuery occurs. Recall that from the proof of the previous claim, if $\mathcal{A}$ wins, $H(Y^*, Z^*, R^*, R^*_Y)$ cannot be defined implicitly during a Prove_x query. On the other hand, NoROQuery implies that $H(Y^*, Z^*, R^*, R^*_Y)$ cannot be defined via an explicit H query either. This means that $H(Y^*, Z^*, R^*, R^*_Y)$ is undefined before $\mathcal{A}$ halts, so it is a random integer in $\mathbf{H}$ in $\mathcal{A}$'s view. Therefore, the probability that $\mathcal{A}$ outputs c^* that is equal to $H(Y^*, Z^*, R^*, R^*_Y)$ — which is part of $\mathcal{A}$'s winning condition — is $1/|\mathbf{H}|$.

In sum, we have proved that $\mathcal{B}$ wins experiment DL as long as $\mathcal{A}$ wins experiment NM and the three "bad events" TwoROValues, ZeroDenominator and NoROQuery do not occur. Therefore,

$$\begin{aligned}\Pr[\mathcal{B} \text{ wins}] &\geq \Pr[\mathcal{A} \text{ wins}] - \Pr[\mathsf{ZeroDenominator}] - \Pr[\mathsf{TwoROValues}] \\ &\quad - \Pr[\mathsf{NoROQuery} \wedge \mathcal{A} \text{ wins}] \\ &\geq \Pr[\mathcal{A} \text{ wins}] - \frac{q(q + 2q_H - 1)}{2p} - \frac{q_H + 1}{|\mathbf{H}|}.\end{aligned}$$

Since $\Pr[\mathcal{B} \text{ wins}]$ negligible as the DL assumption holds for $(\mathbb{G}, p, g)$, we know that $\Pr[\mathcal{A} \text{ wins}]$ is also negligible. This completes the proof. ■

Comparison with the proof in [10]. Our proof bears some similarities to the proof of [10, Theorem 5.6], which shows the non-malleability of ECVRF in the ROM only (and not in the AGM). In particular, our reduction simulates

the oracles Prove_x, HTC and H in the same way as [10, Algorithm 6], and the argument that the reduction's simulation produces an indistinguishable view for the adversary, is essentially the same as the proof of [10, Claim 5.9] The two proofs differ significantly in what the reduction does after the adversary halts: [10] rewinds the adversary which incurs a quadratic security loss, while we avoid this loss by running the adversary only once and relying on the AGM to solve DL.

Further discussion. Just as [10, Theorem 5.6], our non-malleability proof does not use the fact that Y and R_Y are part of the inputs to H. Including Y and R_Y is necessary for other security properties such as uniqueness, which are out of the scope of this work; see [10, Remark 5.11] for further discussion.

Finally, we remark that our proof still holds even if the verification algorithm takes as input group elements of a larger group $\mathbb{E}$ (see Section 2.6). In the context of non-malleability, this gives the adversary the additional ability of outputting $Z^* \in \mathbb{E} \setminus \mathbb{G}$. However, this does not affect the fact that the adversary must make the crucial query in order to win with non-negligible probability, from which the reduction can obtain two expressions of R^* using g and X. (Note that $R^* \in \mathbb{G}$ even if $Z^* \in \mathbb{E} \setminus \mathbb{G}$, as Z^* is only used while computing R_Y^* which is not needed by the reduction.) Therefore, ECVRF still has tight non-malleability even in this case.

References

1. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO 1992*, pages 89–105.
2. B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT 2018*, pages 66–98.
3. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*, pages 186–194.
4. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *CRYPTO 2018*, pages 33–62.
5. G. Fuchsbauer, A. Plouviez, and Y. Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In *EUROCRYPT 2020*, pages 63–95.
6. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP 2017*, pages 51–68.
7. S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák. Verifiable random functions (VRFs), 2022. `https://datatracker.ietf.org/doc/draft-irtf-cfrg-vrf`.
8. S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *FOCS 1999*, pages 120–130.
9. 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. `https://eprint.iacr.org/2017/099`.
10. C. Peikert and J. Xu. Classical and quantum security of elliptic curve VRF, via relative indifferentiability. In *CT-RSA 2023*, pages 84–112.

11. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT 1996*, pages 387–398.
12. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO 2018*, pages 239–252.