Making NSEC5 Practical for DNSSEC

Dimitrios Papadopoulos
HKUST

Duane Wessels
Verisign Labs

Shumon Huque
Salesforce

Moni Naor
Weizmann Institute

Jan Včelák
NS1

Leonid Reyzin
Boston University

Sharon Goldberg
Boston University

ABSTRACT

NSEC5 is a proposed modification to DNSSEC that guarantees two security properties: (1) privacy against offline zone enumeration, and (2) integrity of zone contents, even if an adversary compromises the authoritative nameserver responsible for responding to DNS queries for the zone. In this work, we redesign NSEC5 in order to make it practical and performant. Our NSEC5 redesign features a new verifiable random function (VRF) based on elliptic curve cryptography (ECC), along with a cryptographic proof of its security. This VRF is also of independent interest, as it is being standardized by the IETF and being used by several other projects. We show how to integrate NSEC5 using our ECC-based VRF into DNSSEC, leveraging precomputation to improve performance and DNS protocol-level optimizations to shorten responses. Next, we present the first full-fledged implementation of NSEC5 for both nameserver and recursive resolver, and evaluate performance under aggressive DNS query loads. We find that our redesigned NSEC5 can be viable even for high-throughput scenarios.

KEYWORDS

DNSSEC, verifiable random functions, elliptic curve cryptography, implementation

1 INTRODUCTION

NSEC5 is a new proposal for providing *authenticated denial of existence* for DNSSEC, *i.e.,* for providing authenticated responses to DNS queries ("What is the IP address of `aWa2j3.com`?") for names that do not exist in a zone ("NXDOMAIN: `aWa2j3.com` does not exist in the `.com` zone."). NSEC5 has two key properties.

First, NSEC5 provides *strong integrity*, protecting the integrity of the zone contents even if an adversary compromises the authoritative nameserver (who is responsible for responding to DNS queries for the zone) and steals any secret keys stored on the nameserver.

Second, NSEC5 provides *privacy against offline zone enumeration* [14, 22, 26, 58, 67, 70, 85, 86, 88], an attack in which an adversary makes a small number of online DNS queries and then processes the retrieved records offline in order to learn all the domain names in a zone. An enumerated zone can be "a source of probable e-mail addresses for spam, or as a key for multiple WHOIS queries to reveal registrant data that many registries may have legal obligations to protect" [58] (*e.g.,* per data protection laws [77], [17, pg. 37]). Moreover, zone enumeration can be used to identify routers, servers or other "things" (thermostats, fridges, baby monitors, *etc.*) that could be targeted for attacks, as DNS is increasingly being used to support other Internet protocols [76]. For example, in DNS there are resource types that are used to store host secure shell (SSH) key hashes and other entries that may identify mail servers for a domain. In particular, in the context of the *DNS-Based Authentication of Named Entities (DANE)* protocol, zone enumeration can be used to extract public-key/certificate lists together with the corresponding user identifiers [60].

While today's DNSSEC standard has several mechanisms for authenticated denial of existence, they all either fail to provide strong integrity against a compromised nameserver (*i.e.,* online signing used in NSEC3 White Lies [42] and Minimally-Covering NSEC [87]), or fail to prevent offline zone enumeration (NSEC [18], NSEC3 [58]). In fact, offline zone enumeration is an issue introduced by DNSSEC, and is not a possible attack on legacy DNS. Several publicly available network reconnaissance tools can be used to launch DNSSEC zone-enumeration attacks [9, 14, 26, 67, 70, 85] and the risk of zone enumeration attacks has deterred some deployments of DNSSEC (see *e.g.,* [45]).

The need for online crypto. Somewhat surprisingly, Goldberg *et al.* [47] demonstrated that any mechanism that (a) provides integrity and (b) prevents offline zone enumeration *necessarily* requires the nameserver to perform at least one online cryptographic operation for each negative response. This explains why the only existing DNSSEC mechanism (besides NSEC5) to prevent zone-enumeration attacks has to resort to online signing. However, online signing provides only *weak* integrity (*i.e.,* against network attackers but not compromised nameservers), while NSEC5 provides *strong* integrity (*i.e.,* even when the nameserver is compromised and any secret keys stored on the nameserver are stolen).

Summary of Our Contributions. Goldberg *et al.* [47] gave the first proposal for NSEC5. However, that work focused mostly on the cryptographic aspects of the problem and did not satisfactorily handle the issues that arise in relation to deploying NSEC5 in practice and the complex realities of DNSSEC. We improve three key aspects of that work.

First, we completely redesign the cryptographic mechanism behind NSEC5, basing it on elliptic-curve cryptography rather than on RSA cryptography used in [47]. This redesign aligns with the current push in the DNS community to replace RSA, which is widely used in DNSSEC deployments [16, 81], with elliptic curve cryptography (ECC) [51, 80, 83] in order to have shorter DNSSEC responses at a better security level. Second, we design and optimize the protocol that integrates the NSEC5 crypto into the DNSSEC, taking particular care with wildcard records, which were explicitly ignored in prior work [47, footnote 2]. Third, we present a full-fledged implementation of an authoritative nameserver and recursive resolver

*Parts of this work conducted while the first and third authors were at Verisign Labs and the fifth author was at VCZ.NIC.

that support both RSA- and our new ECC-based NSEC5, and evaluate their performance. This evaluation is necessary in order to understand how the need for on-line cryptography (which we know is necessary to prevent zone enumeration) affects overall DNSSEC perfomance.

Desiderata. Our design focuses on two key desiderata. First, even though NSEC5 necessarily requires the nameserver to perform online cryptographic computations, we want our new ECC-based NSEC5 to be viable even for high-throughput nameserver scenarios.

Second, we want NSEC5 to produce short DNSSEC responses. DNSSEC naturally amplifies DNS responses by including cryptographic keys and digital signatures. Several unfortunate things occur when long DNSSEC responses no longer fit in a single IP fragment [69–71]. Long responses sent over UDP can be fragmented across multiple IP fragments, and thus risk being dropped by a middlebox that blocks IP fragments [79, 84] or being subject to an IP fragmentation attack [49]. Alternatively, the resolver can resend the query over TCP [37, 64], harming performance (due to roundtrips needed to establish a TCP connection) and availability (because some middleboxes block DNS over TCP) [79]. Worse yet, long DNSSEC responses can be used to amplify DDoS attacks [41]. In a DDoS amplification attack, a botnet sends nameservers many small DNS queries that are spoofed to look like they come from a victim machine, and the nameservers respond by pelting the victim machine with many long DNSSEC responses. Long DNSSEC responses increase the volume of traffic that arrives at the victim.

1.1 Results

Design (Sections 2–4). We achieve our desiderata by attacking the problem along two lines: (1) cryptographic design and (2) DNSSEC protocol design.

To redesign the cryptography behind NSEC5, in Section 2 we start from an observation from [65] that NSEC5 can be constructed from any VRF [62]. We construct a fast and practical VRF based on ECC that is also of independent interest (Section 2.3) and is being currently considered for standardization by the IETF. We prove our VRF's security in the random oracle model, using concrete analysis per the formulation of [24]. Our detailed security analysis also allows us to optimize performance. For an $\ell = 128$ bit security level, our ECC-based VRF produces proofs that are 5x shorter and 9x faster than the RSA-based crypto of the original NSEC5 proposal [47].

Section 3 presents NSEC5 and its properties, and describes how it can be based on our ECC-based VRF. Section 4 then re-designs the way DNS interacts with NSEC5, using optimizations that precompute parts of the response (thus resulting in faster query processing), and reducing the number of records in the response (thus resulting in shorter proofs).

NSEC5 Implementation and Performance (Section 5). We implemented an authoritative nameserver and recursive resolver that support both RSA- and ECC-based NSEC5. For the nameserver implementation, we extended Knot DNS 1.6.4 [10], a highly-optimized authoritative implementation. For the recursive resolver, we extended Unbound 1.5.9. one of the most widely used recursive resolver implementations.

Our experimental evaluation demonstrates that our ECC-based NSEC5 can be viable even for high-throughput scenarios. Our nameserver can handle a few tens of thousands of queries per second (64K query/second) on a moderately-sized multi-core server (*i.e.,* 24 threads). This is roughly an order of magnitude larger than the average negative response rate at a single server in the DNS root zone [1]. In fact, our ECC-based NSEC5 nameserver achieves a throughput that is about 2x higher than the only nameserver implementation that prevents offline zone enumeration, and is widely deployed and compliant with the DNSSEC standards (PowerDNS with NSEC3 White Lies [15]). Our NSEC5-ready resolver performs comparably to DNSSEC's existing denial-of-existence mechanisms which, however, fail to simultaneously achieve strong integrity and privacy against zone enumeration.

Response lengths (Section 5.1). We show that our ECC-based NSEC5 responses fit into a single IP fragment, and have lengths that are comparable to ECC versions of the current DNSSEC protocol (*i.e.,* NSEC3 with ECDSA signatures). In fact, ECC-based NSEC5 produces NXDOMAIN responses that are *shorter* than those produced by today's dominant DNSSEC deployment configuration (*i.e.,* NSEC3 with 1024-bit RSA signatures [16, 81]), which has a lower security level!

2 EFFICIENT VERIFIABLE RANDOM FUNCTIONS

A Verifiable Random Function [62] is the public-key version of a keyed cryptographic hash. Only the holder of the secret VRF key can compute the hash, but anyone with the corresponding public key can verify the correctness of the hash. While this cryptographic primitive has been around for almost two decades, recent years have seen its use in a variety of practical applications including NSEC5, end-user public-key transparency [61], and cryptocurrencies [44].

To support NSEC5, we now present a fast and practical ECC-based VRF construction. We carefully prove its security in the random oracle model, using concrete security analysis (in Appendix B) per the formulation of [24], which allows us to optimize performance. Our VRF produces proofs that are 5x shorter and 9x faster than the RSA-based VRF that was implicit in [47], for the same $\ell = 128$ bit security level (see Table 1).

This VRF is of independent interest because of the other applications mentioned above. In fact, a VRF very similar to ours, but with a crucial security flaw, was already suggested in [39]; this flaw migrated into the CONIKS paper and implementation [61], the Google Key Transparency project [2], and the Yahoo! coname project [3]. These have since been corrected as a result of our work [6, 7]. A similar VRF, but without security proofs or exact security analysis that allows us to set lengths, is described in [73, Section 4]. Our VRF is being standardized by IETF [48][1] and was already adopted by others (including the Algorand cryptocurrency [44] and the Yahoo! coname project for providing transparency for end-user keys).

Description of Verifiable Random Function. A VRF comes with a public-key pair (PK, SK); SK is held by the Prover, and PK by the Verifier. The Prover hashes an input α using SK as $\beta = F_{SK}(\alpha)$. This hashing is deterministic, in the sense that F always produces

[1] In August 2017, the Crypto Forum Research Group (CRFG) adopted our VRF Internet draft as a working group document.

the same output β given a pair of inputs (SK, α). The secret key SK is also used to construct a proof π that β is the correct hash output: $\pi = \mathsf{Prove}_{SK}(\alpha)$. All the VRFs we consider allow anyone to deterministically compute β directly from π as $\beta = \mathsf{Proof2Hash}(\pi)$. This is useful for communication-constrained applications like NSEC5, since the Prover can just send π, without sending β. (Note: this 'API' formulation for a VRF—using the Proof2Hash function—is our own.) The proof π allows a Verifier with the public key PK to verify that β is the correct VRF hash of input α under key PK. Thus, the VRF also comes with an algorithm $\mathsf{Ver}(PK, \alpha, \pi)$ that outputs VALID if $\beta = \mathsf{Proof2Hash}(\pi)$ is the correct VRF hash of α under key PK, and INVALID otherwise.

2.1 VRF Security Properties

The VRF's security properties, which we overview here, combine those of a pseudorandom function (PRF) (*e.g.,* HMAC) and an unkeyed cryptographic hash (*e.g.,* SHA-256). Formal definitions are in Appendix B.

Pseudorandomness. VRF pseudorandomness guarantees that β is indistinguishable from random by anyone who does not know the secret key SK. More precisely, suppose that (PK, SK) were generated in a trustworthy manner. Then, for any adversarially-chosen 'target' VRF input α, the VRF hash output β (without its corresponding VRF proof π) is indistinguishable from random for any computationally-bounded adversary who does not know SK. This holds even if the adversary can choose other VRF inputs α' and observe corresponding VRF outputs β' and proofs π'. PRFs have a similar pseudorandomness property but unkeyed cryptographic hash functions (*e.g.,* SHA-256) do not (because an adversary that knows α can trivially compute β (as $\beta = \text{SHA-256}(\alpha)$) and distinguish β from random).

Uniqueness. For any fixed PK and any input α, there is a unique VRF output β that can be proven valid. Uniqueness holds even for an adversarial Prover that knows the VRF secret key SK. More precisely, a computationally-bounded adversary cannot, even given SK, find a target input α, two different VRF hash outputs $\beta_1 \neq \beta_2$, and proofs π_1 and π_2 such that $\beta_i = \mathsf{Proof2Hash}(\pi_i)$ and $\mathsf{Ver}(PK, \alpha, \pi_i)$ outputs VALID, for $i = 1, 2$. Unkeyed cryptographic hash functions (*e.g.,* SHA-256) have a similar property, but PRFs do not.

Collision resistance. Like an unkeyed cryprographic hash function, VRFs need to be collision resistant. Collision resistance must hold even for an adversarial Prover that knows the VRF secret key SK. More precisely, it should be computationally infeasible for an adversary to find two distinct VRF inputs α_1 and α_2 that have the same VRF hash β, even if that adversary knows SK. (PRFs lack such a property, since the party holding the secret key to the PRF can always falsely claim that two inputs hash to the same output.)

2.2 VRF Based on RSA

The original NSEC5 construction [47] was not described in terms of VRFs. However, it actually uses the VRF in Figure 1. The VRF proof is simply a deterministic RSA signature (using a "full-domain hash" construction [23]), and the output is simply the cryptographic hash of the VRF proof. VRF verification amounts to an RSA verification of the proof. For completeness, we prove this is a secure VRF (based on the RSA assumption in the random oracle model) in the extended version of our paper [72].

Keys. Let N be a public RSA modulus, let d be a secret RSA exponent and e be its corresponding public exponent. The public VRF key is (e, N) and the secret VRF key is (d, N).

Proving. To hash input α using the private RSA key (d, N), computing the proof value

$$\pi = (MGF(\alpha))^d \mod N$$

MGF is an IETF-standard cryptographic hash that produces outputs one bit shorter than the RSA modulus [20, Sec. 10.2] (*aka,* a "full domain hash" [23]).

Proof2Hash. Compute the hash value β from π as

$$\beta = H(\pi)$$

where H is a cryptographic hash function (*e.g.,* SHA-256)

Verifying. To verify that π is a valid VRF proof for α, use the public RSA key (e, N) to verify that π is a valid RSA signature on $MGF(\alpha)$, *i.e.,* that $\pi^e = MGF(\alpha) \mod N$.

Figure 1: VRF based on RSA.

Why not build a VRF using ECDSA? At this point, one would naturally wonder why we cannot simply replace the RSA signature in Figure 1 with ECDSA. After all, ECDSA signatures are much shorter than RSA signatures at the same security level. (For instance, ECDSA signatures over 256-bit elliptic curves are just 512 bits long and are understood to have an $\ell = 128$-bit security level, comparable to 3072-bit RSA.)

The problem is that there are multiple valid ECDSA signatures for a given message and public key. With randomized ECDSA signatures, the signature is computed using a random nonce. Even "deterministic" ECDSA [74] fails to provide uniqueness given *only* the public key PK, since the signer derives the nonce from a keyed hash of the message. Because the symmetric key k to this hash is independent of the ECDSA public key PK, the prover could produce a different ECDSA signature just by choosing a different key k, and the verifier would never know the difference. Thus, if ECDSA signatures were used in Figure 1, then the VRF prover could produce any number of valid VRF proofs π for a given input α and public key PK, violating VRF uniqueness.

2.3 VRF Based on Elliptic Curves

The starting point for our VRF based on elliptic curves (ECC) is a construction in [40, 46]. We cannot, however, use the construction of [40] as is. While [40] claimed their construction was also a VRF, they did not formally prove that it achieves the VRF properties from Section 2. In fact, their construction has a critical flaw. Specifically, a malicious prover could trivially violate uniqueness for any input α, producing valid VRF proofs for any arbitrary VRF hash output β by using simple algebra. As explained above, this flaw subsequently migrated to follow-up works [61] and deployed protocols.

Our full VRF construction can be seen in Figure 2 and our formal proof of its security properties in Appendix B. Our construction fixes the flaw of [40],[2] without any downgrade in performance. On the contrary, since we provide a concrete (as opposed to asymptotic)

[2]Our fix is to include h^x in the input to H_3 in Figure 2. We also found and fixed other, smaller flaws; we were able to find them because of our detailed concrete security analysis.

Public parameters. Let q be a prime number, and let G a cyclic group of prime order q with generator g. Because checking membership in G may be expensive, we assume G is a subgroup of some group E such that (1) checking membership in E is easy, and (2) the cofactor $f = |E|/|G|$ is not divisible by q. (G may equal E, in which case $f = 1$.) We assume that q, g, f, G and E are public parameters. Let H_1 be a hash function (modeled as a random oracle) mapping arbitrary-length bitstrings to $G - \{1\}$. Let H_2 be a function that takes the bitstring representation of an element of E and shortens it to the appropriate length; we need a 2ℓ-bit output for ℓ-bit security. Let H_3 be a hash function (modeled as a random oracle) mapping arbitrary-length inputs to ℓ-bit integers.

Keys. The secret VRF key $x \in \{1, \ldots, q-1\}$ is chosen uniformly at random. The public VRF key is $PK = g^x$.

Proving. Given the secret VRF key x and input α, compute the proof π as follows:

(1) Obtain the group element $h = H_1(\alpha)$ and raise it to the power of the secret key to get $\gamma = h^x$.
(2) Choose a secret random value $k \in \{0, \ldots, q-1\}$ (for security k must be (pseudo)random and secret; it can, in particular, be a pseudorandom function of x and α)
(3) Compute $c = H_3(g, h, g^x, h^x, g^k, h^k)$.
(4) Let $s = k - cx \bmod q$.

The proof π is the group element γ and the two exponent values c, s, *i.e.,* $\pi = (\gamma, c, s)$.

Proof2Hash. Given proof $\pi = (\gamma, c, s)$, the VRF output β is computed as $\beta = H_2(\gamma^f)$.

Verifying. Given public key PK, verify that proof $\pi = (\gamma, c, s)$ corresponds to the input α and output β as follows:

(1) Compute $u = (PK)^c \cdot g^s$.
(2) Given input α, hash it to obtain $h = H_1(\alpha)$.
Check that $\gamma \in E$.
Compute $v = \gamma^c \cdot h^s$.
(3) Check that $c = H_3(g, h, PK, \gamma, u, v)$.

Figure 2: The EC-VRF. (Multiplicative group notation.) Appendix B proves its security in the random oracle model, based on the *decisional Diffie-Hellman (DDH)* assumption in group G, which roughly says that h^x looks random given the tuple (g, g^x, h).

security analysis as per the formulation of [24], we can optimize the VRF's parameters. Our design shortens the length of VRF proof π, by truncating value c in Figure 2 so that it is only ℓ bits long (rather than 2ℓ bits long). This results in VRF proofs that are ℓ bits shorter, which is important for communications-constrained applications like NSEC5. Finally, following ideas from [73], our VRF design supports groups where the cofactor f is not 1, so that it can be instantiated over the popular Ed25519 elliptic curve [57] which was recently standardized for use with DNSSEC [80].[3]

High-level description. The idea behind the EC-VRF construction in Figure 2 is best understood as follows. The Prover holds secret key x and the verifier holds public key g^x (where g is a generator of group G). To hash an input α, the prover hashes α into the group G to obtain $h = H_1(\alpha)$, and then raises the result to the power of the secret key to obtain $\gamma = h^x$. The VRF hash output β is $\beta = H_2(\gamma) = H_2(h^x)$. Thus, the Prover can send γ as its VRF proof and anyone can use γ to obtain the VRF output β by using H_2 as the Proof2Hash function. This is not sufficient, however. To satisfy uniqueness, the Prover must convince the Verifier that γ was computed correctly, without revealing the secret key x. Thus, the Prover also attaches a non-interactive zero-knowledge proof (NIZK) that $\gamma = h^x$ and the public key g^x have the same discrete logarithm x base g and h, respectively. This NIZK consists of c and s in Figure 2, and it implements an adaptation of the Chaum-Pedersen discrete-log equality protocol [34].

[3]To avoid unnecessary friction with the IETF standards process, we focus on standardized curves and security assumptions.

Table 1: Proving and verifying time benchmarks for our RSA and EC-based VRFs. Experiments on a MacBook Pro with 2.2 GHz Intel Core i7 and 16 GB DDR3 RAM, averaging of 10^6 runs on random strings of 32 characters. (Standard deviations < 20%)

VRF ciphersuite	security parameter	proof (bits)	proving time (ms)	verifying time (ms)
RSA-VRF (3072)	$\ell = 128$	3072	3.1	0.09
RSA-VRF (2048)	$\ell = 112$	2048	0.7	0.05
EC-VRF (P-256)	$\ell = 128$	641	0.3	0.68

Ciphersuites. Our VRF can be instantiated over any group where the decisional Diffie-Hellman (DDH) problem is hard, including the elliptic curves currently standardized in DNSSEC. We designed our VRF to accommodate groups where the cofactor f is not unity, so it can be used with the Ed25519 curve [57] which has cofactor $f = 8$. Our design also supports the (traditionally used with DNSSEC) NIST P-256 curve [53, Sec. 3]) which has cofactor $f = 1$. We instantiate H_3 as the first ℓ bits output by the SHA-256 hash function, while H_2 is the (untruncated) SHA-256 hash function. The hash-to-curve algorithm H_1 should be public-key-dependent, so that a fresh random oracle is created for each public key (this technique is known as "salting" and forces a brute-force attacker to work hard for each key rather for all keys at once). We show one example of H_1 in Appendix B.1; other approaches, such as Icart's [54], SimpleSWU [32], and Elligator [27], are summarized in [78].

2.4 VRF Performance

Let us set security level $\ell = 128$. This allows us to use the NIST P-256 curve or the Ed25519 curve, which operate in finite field F_p where p is a 256-bit prime, and have a security level of 128 bits [25, 53]. We now compute the length of the VRF proof. Because s is a field element, it is 256 bits long. c can be set to 128 bits for 128-bit security. Meanwhile, γ is a point on the elliptic curve, which can be represented with $256 + 1$ bits using point compression. It follows that the proof π will be $|\pi| = 256+1+128+256 = 641$ bits ($5\ell+1$) for an $\ell = 128$-bit security level. To put this in perspective, achieving the same security level with the RSA-VRF requires 3072-bit RSA, which makes VRF proofs about 5 times longer!

We have implemented the RSA-VRF and EC-VRF (with the NIST P-256 curve) in C. As shown in Table 1, EC-VRF proving is faster than RSA-VRF proving, whereas RSA-VRF has faster verification than EC-VRF. This is great for NSEC5, since the prover (nameserver) may typically have to handle large numbers of requests from multiple verifiers (resolvers).

2.5 VRF Security

We now provide proof sketches of three VRF properties: uniqueness, psuedorandomness, and collision resistance. We define and prove them formally in Appendix B. For the purposes of these sketches, assume $E = G$ and therefore $f = 1$.

Uniqueness. The proof is by contradiction. Suppose an adversary, given the secret key x, can come up with some α and an incorrect VRF output value $\beta_1 \neq H_2([H_1(\alpha)]^x)$ for that α, and a valid proof $\pi_1 = (\gamma_1, s_1, c_1)$ for value β_1. The verification function for the VRF computes $h = H_1(\alpha)$ and

$$u = (g^x)^{c_1} g^{s_1} \qquad v = (\gamma_1)^{c_1} h^{s_1}$$

Now take the logarithm of the first equation base g and the logarithm of the second equation base h, subtract the two resulting equations, and express c_1, to get

$$c_1 \equiv \frac{\log_g u - \log_h v}{x - \log_h \gamma_1} \pmod{q}. \tag{1}$$

Now since $\gamma_1 \neq h^x$ (since β_1 is not the correct output value), the denominator is not zero, and there is exactly one c_1 modulo q that satisfies equation (3) for a given $(g, h, g^x, \gamma, u, v)$, *regardless* of s_1. However, recall that the verifier checks that c_1 is equal to the output of the cryptographic hash function H_3 on input $(g, h, g^x, \gamma, u, v)$. Since H_3 is a random oracle, its output is random, and the probability that it equals the unique value determined by its inputs according to (1) is negligible.[4] Thus, we have arrived at our contradiction.

Pseudorandomness. This follows from the DDH assumption, in the random oracle model. Roughly speaking, the pseudorandomness adversary does not know the secret VRF key x, but must distinguish between between pairs (α, β) where β is the VRF hash output on input α, and pairs (α, r) where r is a random value in the range of H_2. This adversary knows the public values g and g^x, and can easily compute $h = H_1(\alpha)$ for any α. However, by the DDH assumption, h^x looks random even given (g, g^x, h), and so $H_2(h^x)$ is pseudorandom in the range of H_2. We note that pseudorandomness can also be proven from the CDH assumption if H_2 is modeled as a random oracle, but the security reduction will be lossier.

Collision resistance. For a collision to happen, $H_2(h_1^x)$ should equal to $H_2(h_2^x)$ where $h_1 = H_1(\alpha_1)$ and $h_2 = H_1(\alpha_2)$ for some $\alpha_1 \neq \alpha_2$. Since h_1 and h_2 are obtained via random oracle queries and raising to the power x is a permutation, the values being fed to H_2 are random. Assuming random values are not likely to collide under H_2, we get collision resistance.

3 NSEC5 FOR DNSSEC

With DNSSEC, a trustworthy *zone owner* is trusted to determine the set of names (`www.example.com`) present in the zone and their mapping to corresponding values (172.18.216.34). *Nameservers* receive information from the zone owner, and respond to DNS queries for the zone made by *resolvers*. DNSSEC's schemes for authenticated denial of existence reflect tradeoffs between integrity and privacy against offline zone enumeration. For the rest of this section, we assume that the reader is familiar with the DNSSEC standard's existing mechanisms for authenticated denial of existence: NSEC [18], NSEC3 [58], and online signing with NSEC3 White Lies [42]. Readers not familiar with these mechanisms should refer to Appendix A. We now present NSEC5 and its security properties. Sections 3.1–3.2 review prior work, while Section 3.3 is a new contribution.

[4] The birthday paradox does not apply here, so that for a 128-bit security level is suffices to have c be 128 bits long.

3.1 Constructing NSEC5 from VRFs

NSEC5 was introduced in [47, 65], to provide both strong integrity (even when the nameserver is compromised and its keys are stolen) and privacy against offline zone enumeration attacks. In an offline zone-enumeration attack, an adversary makes a small number of online queries to the nameserver to collect NSEC* records, and then learns the names present in the zone via offline dictionary attacks on the (hash) values in the NSEC* records. (See Appendix A.) Table 2 summarizes properties of NSEC5 and compares it to DNSSEC's existing mechanisms for authenticated denial of existence.

Structurally, NSEC5 is very similar to NSEC3, except that we replace the cryptographic hashes used in NSEC3 with the hashes computed by a VRF. We now describe NSEC5, and its three types of DNSSEC records: NSEC5, NSEC5KEY and NSEC5PROOF.

The NSEC5KEY. NSEC5 uses a VRF with its own keys. These keys are distinct from the zone-signing key (ZSK) that is used to compute signatures on DNSSEC records. The secret VRF key is known to both the nameserver and the trusted owner of the zone. Meanwhile, the secret ZSK is only known to the trusted owner of the zone. Finally, resolvers get the public ZSK (in a DNSKEY record), and the public VRF key (in an NSEC5KEY record) using the usual DNSSEC key distribution mechanisms.

Why do we need two separate keys, namely the ZSK (for DNSSEC-signing records) and the VRF key (for NSEC5)? This allows us to separate our two security goals. To achieve strong integrity, we follow the approach in NSEC and NSEC3, and provide the secret ZSK to the trusted zone owner but *not to the untrusted nameserver*. Meanwhile, any reasonable definition of privacy against zone enumeration must trust the nameserver; after all, the nameserver holds all the DNS records for the zone, and thus can trivially enumerate all the records in the zone. For this reason, we will provide the secret VRF key to the nameserver, and use the VRF *only* to deal with zone enumeration attacks.

In [47], cryptographic lower bounds are used to prove the nameserver must *necessarily* have some secret cryptographic key. However, NSEC5 manages to provide integrity even if the secret VRF keys stored on the nameserver are compromised or made public—all that is lost is privacy against zone enumeration. This is contrast to any online signing approach, such as NSEC3 White Lies, where compromising the nameserver's secret key eliminates both integrity and privacy against zone enumeration. This follows because with NSEC3 White Lies nameserver holds the secret ZSK(see Appendix A).

Signing NSEC5 records. The trusted zone owner uses the secret VRF key SK to compute the VRF hashes of all the names present in the zone, and lexicographically orders all the hash values. Each consecutive pair of hashes comprises an NSEC5 record. Each NSEC5 record is signed using the secret ZSK. The NSEC5 records and their associated signatures are provided to the nameserver along with the secret VRF key SK.

Table 2: Properties of NSEC*. Online Signing may, for example, refer to NSEC3 White Lies. Note that [47] proved that it is impossible to provide both privacy and weak/strong integrity without online crypto.

	no online crypto	weak integrity	strong integrity	privacy
legacy DNS	✓	X	X	✓
NSEC or NSEC3 (plain)	✓	✓	✓	X
online signing	X	✓	X	✓
NSEC5	X	✓	✓	✓

Responding with NSEC5 and NSEC5PROOFs. To prove the non-existence of a queried name α, the nameserver uses the secret VRF key SK to obtain the VRF hash proof $\pi = \mathsf{Prove}_{SK}(\alpha)$ and VRF hash output $\beta = \mathsf{Proof2Hash}(\pi)$. The nameserver responds with

(1) an NSEC5PROOF record containing π, and
(2) the precomputed NSEC5 record (and the associated DNSSEC signatures) for the pair of hashes lexicographically before and after β.

NSEC5 is almost identical to NSEC3, except that NSEC3 does not have a 'PROOF' record because resolvers can hash α by themselves. (This is why NSEC3 is vulnerable to offline zone enumeration: because the hash is publicly computable.) Notice also we want the VRF proof π to be short, in order to minimize communication from nameserver to resolver.

Validating. The resolver validates the response by

(1) using the public VRF key PK in the NSEC5KEY record to validate that proof π from the NSEC5PROOF corresponds to the query α, *i.e.,* that $\mathsf{Ver}(PK, \alpha, \pi)$ =VALID,
(2) checking that $\beta = \mathsf{Proof2Hash}(\pi)$ falls strictly between the two hash values in the NSEC5 record, and
(3) using the public ZSK to validate the DNSSEC signatures on the NSEC5 record.

3.2 Properties of NSEC5

Table 2 summarizes the properties of NSEC5.

Online crypto. NSEC5 requires online cryptographic computations for negative responses. (But not for positive responses.) For every query α eliciting a negative response, the nameserver uses the secret VRF key SK to compute the NSEC5PROOF record on the fly; this is why it is important for VRF proof computation to be fast. Note that online signing (*e.g.,* 'NSEC3 White Lies', see Appendix A) also requires online cryptographic computations. This is no coincidence: [47] proved that any solution that both (a) prevents zone enumeration and (b) provides at least weak integrity, must *necessarily* use online cryptography. What is interesting about NSEC5 is that it provides *strong* integrity (*i.e.,* even when the nameserver is malicious or compromised). Meanwhile, online signing provides only weak integrity (*i.e.,* against network attackers but not compromised nameservers).

There may be a concern that online cryptographic computations allow an attacker to perform a denial-of-service attack via CPU saturation. This concern is not unique to NSEC5: it exists for any solution that prevents zone enumeration and provides integrity (*e.g.,* 'NSEC3 White Lies', see Appendix A, and 'Black Lies', see Section 6), because online crypto is required per [47]. Currently, the most frequent distributed denial-of-service attacks against DNS utilize botnets to target bandwidth saturation [35, 63]. Due to the relatively low cost of such attacks, their observed scale is often massive (e.g., more than 1Tbps [66]). Further experimentation would be necessary to determine whether CPU-saturation attacks can significantly expand an attacker's capabilities.

Privacy. An attacker can only enumerate the zone by brute force—by sending an *online* query to the nameserver for each name α that it suspects is in the zone.

Suppose an adversary has collected all the NSEC5 records for the zone, and now wants to enumerate the zone using an offline-dictionary attack that 'cracks' the VRF hashes. The adversary must first hash each entry in his dictionary, and then check if any of the hashed dictionary entries match any VRF hashes in the collected NSEC5 records; if there is a match, the adversary has successfully cracked the VRF hash. However, because the adversary does not know the secret VRF key, by pseudorandomness the VRF hash values are indistinguishable from random values. It follows that the adversary cannot hash any of the entries in its dictionary, and thus cannot perform a offline dictionary attack. A formal security proof of this property is in [65].

Strong integrity. Strong integrity is provided even even if a malicious nameserver, or any other adversary, knows the secret VRF key SK. This is because the untrusted nameserver does not know the secret zone-signing key (ZSK). The idea behind the formal proof (see [65]) of this property is simple. Suppose that the secret VRF key SK used with NSEC5 is made public. Resolvers know the correct public VRF key PK, so the VRF's trusted uniqueness ensures that an adversary (that knows SK) cannot trick resolvers into accepting an incorrect VRF hash output. Then, NSEC5 is essentially the same as (plain) NSEC3: the adversary can correctly hash queries on its own, but cannot forge NSEC* records. Thus, for any name α that is present in the zone, the adversary cannot forge an NSEC5 record that falsely claims that α is absent from the zone. In other words, even if the secret VRF key SK is leaked to an adversary, the security of NSEC5 just downgrades to that of (plain) NSEC3.

3.3 Basing NSEC5 on Elliptic Curves

We now describe how our new ECC-based VRF significantly improves the performance of NSEC5. Concrete performance numbers using our full-fledged NSEC5 implementation are in Section 5. The original NSEC5 construction from [47] uses the RSA-based VRF of Section 2.2. Each precomputed NSEC5 record contains two SHA-256 hash outputs, (each corresponding to β in Figure 1), and one DNSSEC signature. Each NSEC5PROOF, generated on the fly, has one RSA value (π in Figure 1). With 2048-bit RSA, this means that NSEC5PROOFs are $|\pi| = 2048$ bits long, and that the nameserver must perform one RSA signing operation (to generate π) for each negative response. Now consider using our EC-VRF. Each NSEC5 record will once again contain two SHA-256 hash outputs (each corresponding to β in Figure 2) along with a DNSSEC signature. Meanwhile, each NSEC5PROOF record will contain the proof value $\pi = (\gamma, c, s)$. Per Table 1, this means that NSEC5PROOFs that are $|\pi| = 641$ bits long (3x shorter), 2x faster for the nameserver to compute, at a better security level ($\ell = 128$ bits as compared to $\ell = 112$ bits for 2048-bit RSA)!

4 DESIGNING THE DNS PROTOCOL USE OF NSEC5

Our next contribution is to design the DNS protocol for NSEC5. To do this, we must move beyond the clean and idealized model used thus far, where each query ("What is the IP for `example.com`?") elicits either a positive response ("172.18.216.34.") or a negative response ("NXDOMAIN: The name does not exist.") In practice, the behavior of NSEC* is much messier, primarily due to a seemingly-unrelated issue: DNS wildcards [58, Section 7.2.1],[43, 59]. It turns out that the performance of each NSEC* rests heavily on how it deals with wildcards, so careful protocol design is crucial. Thus, we begin by describing how DNS wildcards are handled. Then we present some protocol optimizations that are deployed by NSEC5. Finally, we describe how NSEC5 copes with DNS types and key management in Section 4.2.

Background: Wildcards. A wildcard record maps a set of queries to a particular response. For example, if there is a wildcard record for `*.example.com`, then queries for the records `c.example.com` and `a.b.c.example.com` would all be answered with the value in the wildcard record (*e.g.,* "172.18.216.35").

Consider a simple zone consisting of the following four records: `example.com` A, `bar.example.com` A, `www.example.com` A, and `*.www.example.com` A. Suppose a DNS query is made to the zone for `a.b.c.example.com`. The correct response is NXDOMAIN (*i.e.,* the name does not exist). Why? First, `example.com` is the longest ancestor of the queried name that exists in the zone. In DNS terminology, `example.com` is the *closest encloser* for `a.b.c.example.com` [59]. Next, `*.example.com`—the wildcard child of the closest encloser—is not in the zone. Thus, there is no *wildcard expansion* of the queried record `a.b.c.example.com` and the response is NXDOMAIN.

But how can a nameserver use DNSSEC to securely *prove* the absence of relevant wildcards? First, the nameserver must prove that `example.com` is the closest encloser, by proving:

(1) The presence of the *closest encloser* `example.com`.
(2) The absence of the *next closer* `c.example.com`, the name one label longer than the closest encloser.

(Notice that the next closer is sometimes identical to the queried name, *e.g.,* if we had instead queried for `c.example.com`.) Once this is done, the nameserver must additionally prove:

(3) The absence of `*.example.com`, the wildcard child of the closest encloser.

NSEC3 and wildcards. How does NSEC3 prove the three items above? The middle and last item are easily dealt with, by providing the NSEC3 record proving the *absence* of the name, *i.e.,* that contains a pair of hashes h_1, h_2 such that $h_1 < h(\texttt{name}) < h_2$. But what about proving the *presence* of a name (*i.e.,* the first item)? One way to do this is to provide an NSEC3 record that *matches* the name, *i.e.,* that contains a pair of hashes h_1, h_2 such that $h_1 = h(\texttt{name})$. Thus NSEC3 proves the three items by returning three NSEC3 records [58]:

(1) A NSEC3 record *matching* the closest encloser, *i.e.,* an NSEC3 record with two hash values h_1, h_2 such that $h_1 = h(\texttt{example.com})$.
(2) An NSEC3 record *covering* the next closer, *i.e.,* an NSEC3 record containing two hash values h_1, h_2 such that $h_1 < h(\texttt{c.example.com}) < h_2$.
(3) An NSEC3 record *covering* the wildcard, *i.e.,* an NSEC3 record containing two hash values h_1, h_2 such that $h_1 < h(*.\texttt{example.com}) < h_2$.

Table 3: Performance characteristics of NXDOMAIN responses for NSEC*. NSEC3 WL stands for White Lies. RRSIG records are DNSSEC signatures. σ is the bitlength of a DNSSEC signature, 2ℓ is the bitlength of the hash output in the NSEC3 or NSEC5 record, and $|\pi|$ is the bitlength of an NSEC5PROOF.

	online crypto at nameserver	verifications at resolver	max response length
NSEC	none	2 RRSIGs	2σ
NSEC3	none	3 RRSIGs	$3(\sigma + 4\ell)$
NSEC3 WL	1 RRSIG	3 RRSIGs	$3(\sigma + 4\ell)$
NSEC5	1 NSEC5PROOF	2 RRSIGs 2 NSEC5PROOFs	$2(\sigma + 4\ell + \lvert\pi\rvert)$

Thus, wildcards significantly impact performance: a single query can solicit up to three NSEC3 records. Only two records are needed if the same record matches $h(\texttt{example.com})$ and covers $h(\texttt{c.example.com})$. Indeed, this is *always* true for NSEC, so at most two NSEC records are returned for each query. We summarize the impact on performance in Table 3.

4.1 NSEC5 Protocol Optimizations

Adding the wildcard bit to NSEC5. In [43], Gieben and Mekking observed that wildcards could be dealt with by using just *two* NSEC3 records. Their proposal simply requires a *wildcard bit* to be added to each NSEC3 record. If an NSEC3 record contains the pair of hashes h_1, h_2 where $h_1 = h(\texttt{example.com})$, then the wildcard bit is set if `*.example.com` is present in the zone, and cleared otherwise. This simple trick allows us to eliminate the third NSEC3 record! Instead, we just check that the wildcard bit is cleared on the first NSEC3 record. The wildcard bit was not standardized as part of NSEC3, and has not been deployed [42].

NSEC5 uses the wildcard bit, which has significant impact on response lengths and performance (see Table 3). Every query can elicit a response containing (up to) two NSEC5 records, each including a DNSSEC signature (length σ bits) and two hash values (each of length 2ℓ bits), and up to two NSEC5PROOF records (each of length $|\pi|$ bits). We can therefore estimate the total bitlength of a response as

$$|\text{nsec5}| = 2(\sigma + 4\ell + |\pi|) = 2\sigma + 8\ell + 2|\pi| \quad (2)$$

Adding precomputation to NSEC5. Perhaps the biggest performance challenge with NSEC5 is the need for the nameserver to perform online crypto. We now see how to lower this burden on the nameserver.

First recall that all DNSSEC signatures on NSEC5 records *must* be precomputed. (This is because NSEC5 records are signed by the zone-signing key (ZSK). To preserve strong integrity, the nameserver must not know the secret ZSK.) It is also possible to precompute one of the two NSEC5PROOFs. Specifically, the first NSEC5PROOF and NSEC5 record prove the presence of the closest encloser (*i.e.,* `example.com`) are as follows: (1) The NSEC5 record has two hash values h_1, h_2, where h_1 is the VRF hash of the closest encloser, and (2) the NSEC5PROOF has a proof π that h_1 is a correct VRF hash value. The NSEC5PROOF for h_1 can therefore be precomputed

and cached at the same time as the NSEC5 record. This improves performance, since online crypto is only needed for the second NSEC5PROOF.[5]

4.2 Other Protocol Considerations

NODATA Responses. So far, our exposition assumed that all DNS queries are of the same type: the query contains a domain name (`www.example.com`), and the response contains an IPv4 address ("172.18.216.34"). Actually, this is a query for an *A record*. In practice, there are other query types (*e.g.,* the AAAA record for IPv6 addresses). Suppose the example zone described above receives a AAAA query for `www.example.com`. The zone has an A record for `www.example.com`, but not a AAAA one. Thus, the correct response is NODATA, (*i.e.,* "The name exists, but not for queried type").

Because NSEC5, NSEC3, and NSEC records all have the same structure, they all deal with NODATA responses as follows. Every NSEC* record includes a *type bitmap* [18, 58], containing a bit for each type of DNS record (*e.g.,* A, AAAA, NS, MX). Consider the NSEC* record matching `www.example.com`, *i.e.,* that contains a pair of hash values h_1, h_2 such that h_1 is the hash of `www.example.com`. In our example zone, this NSEC* record has its type A bit set, and its other type bits cleared. This NSEC* record would be used to respond to an AAAA query for `www.example.com`. The resolver would conclude the response is NODATA by checking that the AAAA bit cleared. Notice that NODATA responses always use just one NSEC* record.

Key management. NSEC5KEY records can be distributed in the same way as DNSKEY records. Meanwhile, as discussed in Section 3.2, the nameserver must store the secret VRF key but not the secret ZSK. The secret VRF key is not subject to the same security requirements as a regular DNSSEC secret key (*i.e.,* ZSK), because the damage from a compromised VRF key is the same as the damage from a downloaded zone file; integrity is not damaged. Moreover, an attacker who can break into a nameserver to steal the VRF key can probably also download the zone file, anyway. Moreover, the NSEC5KEY can be rolled over using the same procedure to roll a ZSK [56]: the new NSEC5KEY record is published at the nameserver, then old NSEC5 records are replaced by NSEC5 records computed using the new NSEC5KEY, and finally the old NSEC5KEY is removed.

5 NSEC5 PERFORMANCE EVALUATION

We designed and implemented the two NSEC5 variants (RSA and ECC), extending existing DNS software. For the authoritative nameserver, we extended Knot DNS 1.6.4 and for the recursive resolver we extended Unbound 1.5.9. Our implementation supports the full spectrum of negative responses, (*i.e.,* NXDOMAIN, NODATA, Wildcard, Wildcard NODATA, and unsigned delegation). The authoritative nameserver implements the optimization that precomputes the NSEC5PROOFs matching each NSEC5 record (Section 4.1). We did not introduce additional library dependencies; all cryptographic primitives are already present in OpenSSL v1.0.2j, which is used by both implementations. We implemented our elliptic-curve VRF for the NIST P-256 curve. Overall, we added about 9,000 lines of C code.

We now evaluate the performance of NSEC5 and compare it against (plain) NSEC3 and online signing with NSEC3 White Lies (see Appendix A). We consider response length, query processing time at the recursive resolver and nameserver, and throughput, memory and CPU usage at the nameserver.

Configurations. We tested our Knot DNS nameserver implementation in four configurations:

(1) NSEC3 with 2048-bit RSA signatures (DNSSEC Algorithm 8),
(2) NSEC3 with ECDSA signatures over the NIST P-256 curve (DNSSEC Algorithm 13),
(3) NSEC5 with 2048-bit RSA signatures (RRSIG) and NSECPROOF records,
(4) NSEC5 with ECC using the NIST P-256 curve for both signatures (RRSIG) and NSECPROOFs.

The NSEC3 configurations used 10 hash iterations. (This is a common choice in practice, *e.g.,* at the `.ru` zone.) Finally, we used PowerDNS[6] 4.0.1 in "narrow" mode with BIND back-end to evaluate

(5) NSEC3 White Lies with ECDSA signatures over NIST P-256 (DNSSEC Algorithm 13).

For the recursive resolver, we used our NSEC5-ready extension of Unbound in validating and caching mode.

System. All experiments were executed on a machine with 20X Intel Xeon E5-2660 v3 cores with dual thread support for a total of 40 virtual CPUs, and 256GB RAM, running CentOS Linux 7.1.1503 and OpenSSL 1.0.2j. We would expect a typical second level domain (SLD) to have multiple nameservers of roughly this size, possibly at multiple locations. Because network latency is a common denominator for all our schemes, all experiments were performed with this machine hosting both the nameserver (using 24 threads) and the recursive resolver (using up to 16 threads), each listening to a different port.

Stress testing with "purely negative" query loads. Unless otherwise specified, our measurements use synthetic query loads. We elicit negative (NXDOMAIN) responses by sending queries for names from the zone prepended with a random six-alphanumeric-character sequence. We deliberately chose to stress-test our implementation using this aggressive "purely negative" query load. Importantly, a purely negative query load would typically occur only when a server is subject to a volumetric denial-of-service attack; natural DNS traffic usually elicits both positive responses as well as negative ones (NXDOMAIN) [5].

Zone. We test against a real Alexa-100 SLD zone that consists of about 1000 names.[7] Note that most of our results (except for those for the RAM footprint at the nameserver) are largely agnostic to the choice of zone. This follows because we use worst-case query load

[5] As noted in Table 3, a similar precomputation approach is possible with NSEC3 White Lies. Specifically, the presence of the closest encloser `example.com` and the presence/absence of its wildcard child `*.example.com` are known at the time that the zone is signed. Thus, their corresponding NSEC3 records can be precomputed. This optimization is (sort of) performed by the PowerDNS nameserver, which caches and reuses NSEC3 records generated on-the-fly for the closest encloser and wildcard.

[6] We acknowledge that this is not an apples-to-apples comparison but, to the best of our knowledge, PowerDNS is the only widely-deployed open-source nameserver that supports DNSSEC online signing in an RFC-compliant way. Meanwhile, we chose to base our NSEC5 implementation on the performant Knot DNS nameserver.

[7] We used the only domain in the Alexa 100 that we could completely enumerate because it used DNSSEC with NSEC records. (The rest were unsigned, or did not use NSEC records.)

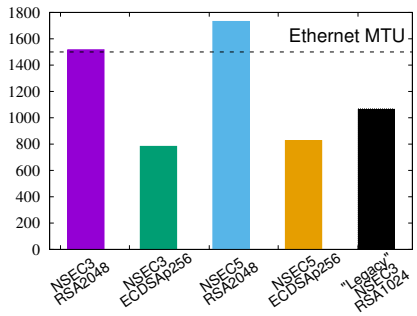

Figure 3: Average length for a single NXDOMAIN response (standard deviation < 1%).

of purely negative traffic, which eliminates the effect of caching, and therefore the size of the zone itself has little impact on performance.

5.1 Response Lengths

We want DNSSEC responses to be short enough to fit into a single IP fragment and to limit DDoS amplification (Section 1). We find that NSEC5-ECC response lengths are comparable to NSEC3 with ECDSA, and *shorter* than today's dominant deployment configuration (NSEC3 with 1024-bit RSA).

Figure 3 shows the average response size for 100,000 NXDOMAIN responses for our four Knot DNS configurations. When RSA is used, both NSEC5 (at 1731 bytes, on average) and NSEC3 (1517 bytes) do not fit in a 1500-byte IP fragment (Ethernet MTU). Meanwhile, ECC-based NSEC5 is much shorter (827 bytes, on average), which is comparable to ECC-based NSEC3 (783 bytes).

Comparison to "legacy" NSEC3. Modern cryptographic recommendations mandate a security level of at least 112 bits [21]. Despite these recommendations, NSEC3 only supports (outdated) SHA1 as its hash function [58], for an (outdated) security level of $\ell = 80$ bits. (NSEC5 records use $2\ell = 256$-bit hash outputs, for a $\ell = 128$-bit security level.) Also, most domains deploying DNSSEC still use 1024-bit RSA ($\sigma = 1024$ bits) [16, 81], for an (outdated) 80-bit security level [21]. NSEC3 with 1024-bit RSA has an average response length of 1069 bytes. This is about 29% *longer* than ECC-based NSEC5, which also has a much stronger security level ($\ell = 128$ versus $\ell = 80$ bits)!

5.2 Nameserver Performance

Both NSEC5 and NSEC3 White Lies prevent offline zone enumeration by requiring online public-key crypto computations at the nameserver. (See Table 2.) We therefore compare their performance at the nameserver, and find that our ECC-based NSEC5 implementation (extending Knot DNS) is *faster* than PowerDNS's implementation of NSEC3 White Lies.

Processing time per query. To measure the time it takes to process a query at the authoritative nameserver, we ran 100,000 sequential queries, each eliciting an NXDOMAIN response. To fairly compare across implementations, we report round-trip time as observed by the query issuer. Plain NSEC3 (with RSA-2048 and ECDSA-P256) uses precomputed responses; as such, the nameserver can respond to queries in approximately 0.1ms on average (standard deviation ≈ 16%). Meanwhile NSEC5 and NSEC3 White Lies use online crypto, therefore process queries more slowly. RSA-based NSEC5 takes 1.93ms on average, while ECC-based NSEC5 presents a 2.3x speedup, for an average query processing time of 0.81ms (standard deviation < 11% for both of them). This is *faster* than the 1.12ms query processing time for the PowerDNS implementation of NSEC3 White Lies (standard deviation ≈ 19%).

Throughput with purely negative traffic. Next, we consider aggregate query throughput. We used Dnsperf 2.1.1 [13], a popular open-source DNS performance evaluation tool, to issue negative queries at fixed rates from 1K to 128K queries per second (qps). Figure 4-(left) presents throughput on a logarithmic scale.

Plain NSEC3 does not use online cryptographic computations, and so throughput scales easily to 128 Kqps and beyond. (Of course, plain NSEC3 is also vulnerable to offline zone enumeration attacks.) The remaining schemes do use online crypto computations. RSA-based NSEC5 plateaus earliest—the nameserver cannot cope with a query rate greater than about 20 Kqps. Turning to elliptic-curve configurations, PowerDNS's NSEC3 White Lies plateaus at about 32 Kqps, while our ECC-based NSEC5 improves on this to almost 64 Kqps. This 2x improvement follows from differences in the Knot DNS and PowerDNS implementations, which is also in line with benchmark results of [11]. Our NSEC5-ECC throughput results should be well above the needs of most zone operators. To put this in context, the A root server operator [1] reports an average negative query load per server that is roughly one order of magnitude smaller. (On July 7, 2017 the total number of NXDOMAIN responses (RCODE 3) that day is 2640833191 split across 5 servers for an average of 6113 queries/second/server.)

Throughput with mixed traffic. In practice, throughput should be even higher, because normal traffic should elicit positive responses (signed A, AAAA, MX, *etc.*, records), which are precomputed, in addition to NXDOMAIN responses. To better demonstrate this, we also tested the ECC-based NSEC5 nameserver at a steady rate of 32 Kqps, using only 4 threads (which makes saturating the nameserver easier). In this case, when fewer than 50% of responses were NXDOMAIN, throughput remained steady at 32 Kqps. Meanwhile, throughput peaked at 13 Kqps with purely NXDOMAIN traffic.

CPU utilization. For a 32 Kqps query load of purely NXDOMAIN traffic, NSEC5 has a CPU utilization (≈ 60%) that is only slightly higher than that of plain NSEC3 (≈ 50%) and better than that of NSEC3 White Lies (with PowerDNS) (≈ 85%). Details are in the extended version [72].

Memory footprint. Table 4 considers the memory footprint at the authoritative nameserver, once the zone is loaded. Because our test SLD zone had only 1000 records, we repeated this experiment for the `.name` TLD, which has about 460, 000 records. We see that ECC generally has a much smaller memory footprint than RSA. NSEC5 also takes up more space than plain NSEC3 because: (i) NSEC5PROOFs are precomputed and cached to optimize performance (Section 4.1), and (ii) NSEC5 records use 256-bit hash values, while NSEC3 uses (outdated, insecure) 160-bit SHA1 hash values. Finally, the overhead for NSEC3 White Lies is only approximately 36MB, as NSEC3 records are computed on the fly.

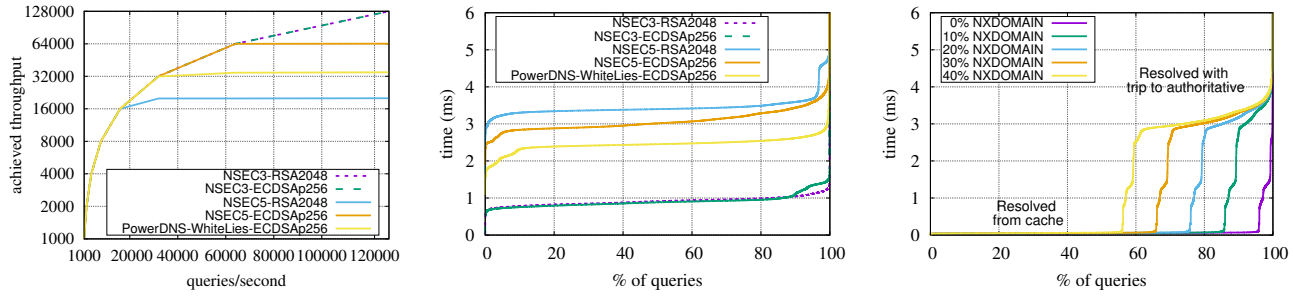

Figure 4: (left) Throughput at the authoritative nameserver under stable query rate when all queries result in NXDOMAIN responses. Overall per query processing time at the recursive resolver and nameserver for (center) NXDOMAIN response across configurations, (right) ECC-based NSEC5 under positive & NXDOMAIN traffic.

Table 4: Memory footprint (MB) at the authoritative nameserver.

	unsigned DNS	NSEC3 RSA2048	NSEC3 ECDSAp256	NSEC5 RSA2048	NSEC5 ECDSAp256	PowerDNS-WL ECDSAp256
tested SLD	18.1	49.3	43.9	64	53.3	18.6
.name TLD	108.3	417.2	254.7	634.1	492.2	144.4

5.3 Recursive Resolver Performance

NSEC3 and NSEC5 both require recursive resolvers to perform public-key crypto verifications (Table 3). We therefore find that query processing times at the recursive resolver for our RSA- and ECC-based NSEC5 implementations are comparable to those of NSEC3.

Overall per-query processing time. Figure 4-(center) reports the overall query processing time per NXDOMAIN response, as observed by a stub resolver. This includes the processing time both at the recursive resolver (which verifies DNSSEC responses) and at the authoritative nameserver (with serves or generates responses). We set up the stub resolver, recursive resolver, and nameserver on our single machine. Our query load was 100,000 sequential unique queries, each eliciting an NXDOMAIN response from the nameserver.

Figure 4-(center) shows that plain NSEC3, NSEC3 White Lies, and NSEC5 all have processing times of the same order of magnitude. This follows because they all require public-key crypto verifications at the recursive resolver. Overall processing time for plain NSEC3 is fastest (1ms on average, standard deviation ≈ 12%); again, this follows because plain NSEC3 does not require online crypto at the nameserver. Of the three configurations that use online crypto at the nameserver to prevent zone enumeration, RSA-based NSEC5 takes the longest (3.4ms on average), followed by NSEC5-ECC (3.1ms on average) and NSEC3 White Lies using PowerDNS (2.4ms on average). Standard deviation < 8% across the three of them.

Mixed traffic. The average query processing time is likely to be faster in practice, since real DNSSEC traffic contains positive responses (*e.g.,* signed A records) as well as NXDOMAIN responses. To highlight this, Figure 4-(right) shows the overall query processing time for ECC-based NSEC5, when handling traffic containing both positive and NXDOMAIN responses. Positive queries were sampled from the zone according to a Zipf distribution, which has been shown to be a good fit for DNS query distributions [55]. Naturally, NSEC5 only affects performance for negative queries; everything else is validated from cache in minimal time.

Validation time. In the extended version of our paper [72], we show that our recursive resolver can validate RSA-based NSEC5 responses (0.5ms) faster than ECC-based NSEC5 responses (1.2ms). (This is natural: RSA verification is known to be faster than ECDSA verification.) We also find that ECC-based NSEC5 validation is slower than PowerDNS's NSEC3 White Lies with ECDSA (0.5ms). The slowdown is due to (1) parsing the NSEC5 response, (2) fetching the NSEC5KEY from cache, and (3) a performance gap between our (unoptimized) ECC-based VRF verification and the highly-optimized OpenSSL verification of ECDSA.

Remark: Speedups with Ed25519? Our implementation uses the NIST P-256 elliptic curve. However, the literature suggests that computational speedups are possible by moving from P-256 to the Ed25519 [80] elliptic curve, which we leave for future work.

6 NSEC5 VS. RECENT INNOVATIONS

We now consider the relationship between NSEC5 and some recent DNS innovations. "Black Lies" [82] is a (concurrent, not standardized) NSEC* proposal. It is an online-signing solution that answers each negative query with an NODATA response, even if the "correct" response is NXDOMAIN. For example, suppose the simple zone from Section 4 receives an AAAA query for `a.example.com`. The Black Lies response is a single NSEC record matching `a.example.com`, with its AAAA type bit cleared, that is generated and signed on the fly. To prevent zone enumeration, the second name in the NSEC record is the immediate lexicographic successor of query, *i.e.,* `\000.a.example.com`. Responses are short because only one NSEC record is required for a NODATA response (see Section 4.2). Black Lies comes with some caveats. Most importantly, it fails to provide strong integrity as it requires the nameserver to know the secret zone-signing key (ZSK). Moreover, because Black Lies gives a NODATA response when the "correct" response is NXDOMAIN, it is not compatible with the performance optimization of

RFC8020 [30]. Also, Black Lies thwarts any diagnostic or security tool (*e.g.,* [38, 75]) that uses NXDOMAIN responses to infer that a name definitely does not exist in the zone.

7 THE TRANSITION TO NSEC5

The DNS community had to handle issues related with protocol transitions in the past. First, the NSEC3 specification [58] came out after the earliest deployments of DNSSEC [68], and so resolvers and nameservers had to transition from NSEC to NSEC3 [58, Section 10.4]. Second, there is currently a proposal to transition from RSA to ECDSA signatures over the NIST P-256 elliptic curve [83]. Third, a desire to avoid NIST-specified curves [28] and to have short responses, is motivating the community to consider transitioning to digital signatures over Edwards elliptic curves [80, 89]. Fourth, there is also the DPRIVE initiative that seeks to add confidentiality to DNS transactions, to mitigate concerns surrounding pervasive network monitoring [4]. Given that other transitions may be on the horizon, this might also be a good time to consider transitioning to NSEC5.

The mechanics of the transition. We believe that a transition to NSEC5 can be accomplished similarly to the transition to NSEC3. DNSSEC records have an *algorithm number* that specifies the cryptographic algorithms they use (*e.g.,* 5 specifies RSA signatures with SHA1 hashing [8]). To transition to NSEC3, two new algorithm numbers were introduced—6:DSA-NSEC3-SHA1 and 7:RSASHA1-NSEC3-SHA1. (Once the transition period ended, subsequent DNSSEC algorithm numbers (8,10, 12, *etc.*) implied support of NSEC3.) Per [19, Sec 5.2], resolvers that did not support NSEC3 ignored DNSSEC records with algorithms 6 or 7, and either 'hard failed' (*i.e.,* rejected the response) or 'soft failed' (*i.e.,* accepted the response) depending on their local policies. Algorithm numbers could also be used to transition to NSEC5. There are two ways [56, Sec 4.1.4] to transition to a new algorithm number.

1. Conservative approach. The nameserver simultaneously supports both algorithms. Thus, the nameserver answers each query with a DNSSEC response has records for both the old and the new algorithm number. The resolver can validate the response if recognizes at least one algorithm. The downside is that DNSSEC responses contain twice as many keys and signatures.

2. Liberal approach. The nameserver stops serving responses with the old algorithm, and uses the new algorithm instead. The downside is that resolvers that do not support the new algorithm number will treat the zone as unsigned [19, Sec 5.2]. Thus, the liberal approach is unlikely to be used until many resolvers support the new algorithm number.

There are several reasons why the liberal approach seems right for NSEC5. First, it does not blow up the length of DNSSEC responses. Secondly, and more importantly, a zone that simultaneously supports both NSEC3 and NSEC5 will not reap the security benefits of NSEC5. If (plain) NSEC3 is supported in parallel with NSEC5, then offline zone enumeration is possible by collecting the NSEC3 records (this also suggests that algorithm negotiation [50] may be less helpful in a transition to NSEC5—a zone-enumeration attacker can simply negotiate to speak NSEC3). If online signing (*e.g.,* NSEC3 White Lies) is supported in parallel with NSEC5, then the nameserver must hold the secret ZSK key and NSEC5 loses its strong integrity guarantees. On the other hand, this approach is unlikely to be used in a transition until a majority of resolvers support NSEC5. However, given resolvers might soon be upgraded to add support for Edwards curves, now might also be a good time to consider adding support for NSEC5.

8 CONCLUSION

In [47], Goldberg et al. proved that providing integrity while preventing offline zone enumeration *necessarily* requires the nameserver to perform online public-key crypto computations for each negative query. While this seems expensive, we demonstrate that our ECC-based NSEC5 nameserver implementation can be viable even for high-throughput scenarios. In Section 5.2 we found that it supports a throughput of 64, 000 negative queries per second (qps) on a moderately-sized server with 24 threads on 40 virtual cores. This is about 2x the throughput of the only implementation of RFC-compliant online signing that is widely deployed and publicly available (PowerDNS's implementation of NSEC3 White Lies). A throughput of 64 Kqps should be well above the needs of most zone operators—even public statistics from the A-root operator [1] indicate an average negative query load about one order of magnitude smaller per server. Without access to proprietary statistics regarding corporate second-level-domains, it is not easy to estimate their throughput requirements. Nevertheless, this 64 Kqps throughput is achieved even with purely negative traffic (rather that normal traffic, *i.e.,* a mix of positive and negative queries) and a single server (rather than a cluster of nameservers, a more common configuration).

Thus, we believe that NSEC5 can be a practical solution for zones that care about protecting sensitive information (names of hosts, servers, routers, IoT devices, DANE certificates [52], *etc.*) from offline zone enumeration attacks. Currently, such zones must use NSEC3 White Lies, which requires them to trust their nameservers with the sensitive secret zone-signing key. We have shown that NSEC5 provides comparable/improved performance to NSEC3 White Lies (Section 5), while protecting integrity even in the face of a compromised nameserver (Table 2). That said, operators that don't care about zone enumeration should use plain NSEC3.

Transitioning DNSSEC to a new algorithm has never been easy. Nevertheless, DNSSEC might be faced with several algorithm transitions on the horizon, including a transition from RSA to ECDSA [83] to EdDSA [80, 89]. Thus, we believe that the time is right to consider a transition to NSEC5. Indeed, the results presented in this paper have helped motivate practitioners to take a closer look at NSEC5, as evidenced by the support of engineers at several major DNS providers, recent presentations at IETF98, DNS-OARC, DPRIVE, and the IETF draft specifying NSEC5.

ACKNOWLEDGEMENTS

We thank innumerable DNS practitioners for pushing us to develop a more performant version of NSEC5. We thank David C. Lawrence for his ongoing work on the NSEC5 project and Asaf Ziv, Sachin Vasant, Ondrej Sury and Tomofumi Okubo for earlier work on NSEC5. We thank Trevor Perrin for helpful suggestions. This research was supported, in part, by NSF grants 1012798, 1012910, 1350733 and 5245250 and a gift from Verisign Labs.

REFERENCES

[1] A Root server raw data. http://a.root-servers.org/raw-data/index.html (Accessed: 1/4/2017).

[2] Google Key Transparency: A transparent and secure way to look up public keys. https://github.com/google/keytransparency/.

[3] Yahoo! Coname Project. https://github.com/yahoo/coname/.

[4] DNS PRIVate Exchange Working Group Charter (DPRIVE), 2015. https://datatracker.ietf.org/doc/charter-ietf-dprive/.

[5] A-root Query Volume. http://a.root-servers.org/metrics/index.html, January 6 2016.

[6] A CONIKS implementation in Golang: Issue 175: Uniqueness of VRF is violated. https://github.com/coniks-sys/coniks-go/issues/175, April 2017.

[7] Google Key Transparency Project: Issue 567: Uniqueness of VRF is violated. https://github.com/google/keytransparency/issues/567, April 2017.

[8] IANA Domain Name System Security (DNSSEC) Algorithm Numbers http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml, 2017.

[9] Kali Tools: DNSRecon. http://tools.kali.org/information-gathering/dnsrecon, 2017.

[10] Knot DNS. https://www.knot-dns.cz/, 2017.

[11] Knot DNS: Benchmark. https://www.knot-dns.cz/benchmark/, 2017.

[12] Nmap: dns-nsec-enum. https://nmap.org/nsedoc/scripts/dns-nsec-enum.html, 2017.

[13] Nominum Measurement Tools. http://www.nominum.com/measurement-tools/, 2017.

[14] nsec3map: John the Ripper plugin. https://github.com/anonion0/nsec3map, 2017.

[15] PowerDNS. https://www.powerdns.com/, 2017.

[16] Verisign Labs SecSpider: Global DNSSEC deployment tracking, 2017. http://secspider.verisignlabs.com/.

[17] B. Aitken. Interconnect communication MC / 080:DNSSEC Deployment Study. http://stakeholders.ofcom.org.uk/binaries/internet/domain-name-security.pdf, 2011.

[18] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *RFC 4034: Resource Records for the DNS Security Extensions.* IETF, 2005.

[19] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *RFC 4035: Protocol Modifications for the DNS Security Extensions.* IETF, 2005.

[20] J. S. B. Kaliski. *RFC 2437: PKCS #1: RSA Cryptography Specifications, Version 2.0.* IETF, 1998.

[21] E. Barker and Q. Dang. Recommendation for Key Management - Part 3 Application-Specific (Revised). NIST Special Publication 800-57, January 2015.

[22] J. Bau and J. C. Mitchell. A security evaluation of DNSSEC with NSEC3. In *NDSS*, 2010.

[23] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.

[24] M. Bellare and P. Rogaway. The exact security of digital signatures - How to sign with RSA and Rabin. In *EUROCRYPT*, pages 399–416, 1996.

[25] D. J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *PKC*, pages 207–228. 2006.

[26] D. J. Bernstein. NSEC3 Walker. http://dnscurve.org/nsec3walker.html, 2011.

[27] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *ACM CCS*, pages 967–980, 2013.

[28] D. J. Bernstein, T. Lange, and R. Niederhagen. Dual EC: A standardized back door. In *The New Codebreakers*, pages 256–281. Springer, 2016.

[29] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *ASIACRYPT*, pages 514–532, 2001.

[30] S. Bortzmeyer and S. Huque. *RFC 8020: NXDOMAIN: There Really Is Nothing Underneath.* IETF, 2005.

[31] C. Boyd, P. Montague, and K. Nguyen. Elliptic Curve Based Password Authenticated Key Exchange Protocols. In *Information Security and Privacy*, volume 2119 of *LNCS*, pages 487–501. 2001.

[32] E. Brier, J. Coron, T. Icart, D. Madore, H. Randriam, and M. Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2010.

[33] M. Chase and A. Lysyanskaya. Simulatable VRFs with Applications to Multi-theorem NIZK. In *CRYPTO'07*, pages 303–322, 2007.

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

[35] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet detection by monitoring group activities in DNS traffic. In *Seventh International Conference on Computer and Information Technology (CIT 2007), October 16-19, 2007, University of Aizu, Fukushima, Japan*, pages 715–720. IEEE Computer Society, 2007.

[36] J. Coron. On the Exact Security of Full Domain Hash. In *CRYPTO*, pages 229–235, 2000.

[37] J. Damas, M. Graff, and P. Vixie. *RFC 6891: Extension Mechanisms for DNS (EDNS(0)).* IETF, 2013.

[38] C. Deccio. DNSviz: a tool for visualizing the status of a DNS zone. http://dnsviz.net/, 2010.

[39] M. Franklin and H. Zhang. Unique ring signatures: A practical construction. Technical Report 2012/577, ePrint Cryptology Archive, 2012 (updated 2017).

[40] M. Franklin and H. Zhang. Unique ring signatures: A practical construction. In *FC*, pages 162–170. Springer, 2013.

[41] S. Gibson. Distributed Reflection Denial of Service (DrDoS) Attacks. Technical report, Gibson Research, 2002.

[42] R. Gieben and W. Mekking. *RFC 7129: Authenticated Denial of Existence in the DNS.* IETF, 2014.

[43] R. Gieben and W. Mekking. draft-gieben-nsec4-00:DNS Security (DNSSEC) Authenticated Denial of Existence, 2015.

[44] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*, pages 51–68, 2017.

[45] D. K. Gilmore. Meeting minutes, DNSOP meeting at IETF98. https://www.ietf.org/proceedings/98/minutes/minutes-98-dnsop-00.txt, March 2017.

[46] E. Goh and S. Jarecki. A Signature Scheme as Secure as the Diffie-Hellman Problem. In *EUROCRYPT*, pages 401–415, 2003.

[47] S. Goldberg, M. Naor, D. Papadopoulos, L. Reyzin, S. Vasant, and A. Ziv. NSEC5: provably preventing DNSSEC zone enumeration. In *NDSS*, 2015.

[48] S. Goldberg, D. Papadopoulos, and J. Vcelak. draft-goldbe-vrf: Verifiable Random Functions, 2017.

[49] A. Herzberg and H. Shulman. Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org. In *IEEE CNS*, pages 224–232, 2013.

[50] A. Herzberg and H. Shulman. Negotiating DNSSEC Algorithms over Legacy Proxies. In *CANS*, pages 111–126, 2014.

[51] A. Herzberg and H. Shulman. Cipher-Suite Negotiation for DNSSEC: Hop-by-Hop or End-to-End? *Internet Computing, IEEE*, 19(1):80–84, 2015.

[52] P. Hoffman and J. Schlyter. *RFC 6698: The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSAC.* IETF, 2012.

[53] P. Hoffman and W. Wijngaards. *RFC 6605: Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC.* IETF, 2012. https://tools.ietf.org/html/rfc6605.

[54] T. Icart. How to hash into elliptic curves. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316. Springer, 2009.

[55] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS performance and the effectiveness of caching. In *ACM IMW*, pages 153–167, 2001.

[56] O. Kolkman, W. Mekking, and R. Gieben. *RFC 6781: DNSSEC Operational Practices, Version 2.* IETF, 2012.

[57] A. Langley, M. Hamburg, and S. Turner. *RFC 7748: Elliptic Curves for Security.* IETF, 2016.

[58] B. Laurie, G. Sisson, R. Arends, and D. Blacka. *RFC 5155: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence.* IETF, 2008.

[59] E. Lewis. *RFC 4592: The Role of Wildcards in the Domain Name System.* IETF, 2006.

[60] A. Mankin and S. Huque. DNS Privacy Overview. DNS-OARC Fall Workshop. https://indico.dns-oarc.net/event/24/contributions/357/attachments/330/586/dns-privacy.pdf, 2015.

[61] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman. CONIKS: Bringing Key Transparency to End Users. In *USENIX Security*, pages 383–398, 2015.

[62] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable Random Functions. In *FOCS*, pages 120–130, 1999.

[63] J. Mirkovic and P. L. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *Computer Communication Review*, 34(2):39–53, 2004.

[64] P. Mockapetris. *RFC 1035: Domain Names - Implementation and Specification.* IETF, 1987. https://tools.ietf.org/html/rfc1035.

[65] M. Naor and A. Ziv. Primary-secondary-resolver membership proof systems. In *TCC*, pages 199–228. 2015.

[66] L. H. Newman. What we know about Friday's massive East Coast Internet outage, 2016. https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/.

[67] NLNetLabs. ldns. http://git.nlnetlabs.nl/ldns/tree/examples/ldns-walk.c, 2017.

[68] E. Osterweil, D. Massey, and L. Zhang. Observations from the DNSSEC Deployment. In *NPSec*, 2007.

[69] E. Osterweil, D. Massey, and L. Zhang. Availability problems in the DNSSEC deployment, 2009. http://irl.cs.ucla.edu/talks/2009-05-RIPE-PMTU.pptx.

[70] E. Osterweil, D. Massey, and L. Zhang. Deploying and monitoring DNS security (DNSSEC). In *ACSAC*, pages 429–438, 2009.

[71] E. Osterweil, M. Ryan, D. Massey, and L. Zhang. Quantifying the operational status of the DNSSEC deployment. In *ACM IMC*, pages 231–242, 2008.

[72] 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.

[73] T. Perrin. The XEdDSA and VXEdDSA signature schemes (rev. 1, 2-16-1020), 2016. https://signal.org/docs/specifications/xeddsa/xeddsa.pdf.

[74] T. Pornin. *RFC 6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).* IETF, 2013.
[75] Root Server System Advisory Committee (RSSAC). RSSAC002: RSSAC Advisory on Measurements of the Root Server System. Technical report, ICANN, November 2014. https://www.icann.org/en/system/files/files/rssac-002-measurements-root-20nov14-en.pdf.
[76] S. Rose and A. Nakassis. Minimizing information leakage in the DNS. *IEEE Network*, 22(2):22–25, 2008.
[77] M. Sanz. DNSSEC and the Zone Enumeration. European Internet Forum: http://www.denic.de/fileadmin/public/events/DNSSEC_testbed/zone-enumeration.pdf, 2004.
[78] S. Scott, N. Sullivan, and C. Wood. *draft-sullivan-hash-to-curve-01: Hashing to Elliptic Curves: Internet Draft.* IETF, 2018.
[79] M. Sivaraman, S. Kerr, and L. Song. *draft-muks-dns-message-fragments-00: DNS message fragments.* IETF, 2015.
[80] O. Sury and R. Edmonds. *draft-ietf-curdle-dnskey-ed25519: Ed25519 for DNSSEC.* IETF, 2016.
[81] L. Valenta, S. Cohney, A. Liao, J. Fried, S. Bodduluri, and N. Heninger. Factoring as a Service. In *FC*, pages 321–338, 2016.
[82] F. Valsorda and O. Gudmundsson. *draft-valsorda-dnsop-black-lies: Compact DNSSEC Denial of Existence or Black Lies (expired).* IETF, 2016.
[83] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. Making the Case for Elliptic Curves in DNSSEC. *ACM CCR*, 45(5):13–19.
[84] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. DNSSEC and its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *ACM IMC*, pages 449–460, 2014.
[85] M. Wander. nsec3breaker. https://www.vs.uni-due.de/trac/dnssec, 2016.
[86] M. Wander, L. Schwittmann, C. Boelmann, and T. Weis. GPU-Based NSEC3 Hash Breaking. In *IEEE NCA*, 2014.
[87] S. Weiler and J. Ihren. *RFC 4470: Minimally Covering NSEC Records and DNSSEC On-line Signing.* IETF, 2006.
[88] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang. Deploying Cryptography in Internet-Scale Systems: A Case Study on DNSSEC. *IEEE TDSC*, 8(5):656–669, 2011.
[89] D. York, O. Sury, P. Wouters, and O. Gudmundsson. *draft-york-dnsop-deploying-dnssec-crypto-algs: Observations on Deploying New DNSSEC Cryptographic Algorithms.* IETF, 2016.

A BACKGROUND: TODAY'S DNSSEC

With DNSSEC, a trustworthy *zone owner* determines the set of names (`www.example.com`) present in the zone and their mapping to corresponding values (172.18.216.34). *Nameservers* receive information from the zone owner, and respond to DNS queries for the zone made by *resolvers*. DNSSEC's schemes for authenticated denial of existence reflect tradeoffs between integrity and privacy against offline zone enumeration. We describe each scheme below:

NSEC (RFC 4034 [18]). The NSEC record is defined as follows. The trusted owner of the zone prepares a lexicographic ordering of the names present in a zone, and uses the secret *zone signing key (ZSK)* to sign a record containing each consecutive pair of names. The precomputed NSEC records are then provided to the nameserver. Then, to prove the non-existence of a name (`x.example.com`), the nameserver returns the NSEC record corresponding to the pair of existent names that are lexicographically before and after the non-existent name (`w.example.com` and `z.example.com`), with its associated DNSSEC signatures. NSEC provides **strong integrity**—it not only protects against network attackers that attempt to alter DNSSEC responses, but is also robust to a malicious nameserver. This is because NSEC records are precomputed and signed by the trusted owner of the zone, and so the nameserver does not need to know the secret ZSK in order to produce a valid NSEC record. Without the secret ZSK, a malicious nameserver cannot sign bogus DNSSEC responses. On the other hand, NSEC is vulnerable to trivial zone enumeration attacks: N *online* queries to the nameserver suffice to enumerate all N names in the zone. Several network reconnaissance tools use NSEC records to enumerate DNS zones [9, 12, 67, 70].

NSEC3 (RFC 5155 [58]). NSEC3 is meant to raise the bar for zone enumeration attacks. The trusted owner of the zone cryptographically hashes all the names present in the zone using SHA1, lexicographically orders all the hash values, and uses the secret ZSK to sign a NSEC3 record containing every consecutive pair of hashes. To prove the non-existence of a name, the nameserver returns the precomputed NSEC3 record (and associated DNSSEC signatures) for the pair of hashes lexicographically before and after the *hash* of the non-existent name.

Since NSEC3 records are precomputed, it also provides strong integrity. However, [26, 86] demonstrated (and RFC 5155 [58, Sec. 12.1.1] acknowledged) that hashing does not eliminate zone enumeration. To enumerate a zone that uses NSEC3, the adversary again makes a number of *online* queries to the nameserver to collect all the NSEC3 records, and then uses an *offline* dictionary attack to crack the hash values in the NSEC3 records, thus learning the names present in the zone. These offline attacks will only become faster as new tools come online [12, 14, 85] and technologies for fast hashing continue to improve (*e.g.,* GPUs [86], ASICs).

Online signing with NSEC3 White Lies (RFC 7129 [42]). Neither NSEC nor NSEC3 prevent zone enumeration. As a result, the DNS community introduced a radically different approach that prevented zone enumeration at the cost of sacrificing strong integrity. DNSSEC *online signing* requires the nameserver to hold the secret zone-signing key (ZSK), and to use it to generate NSEC3 responses on the fly. Crucially, online signing does not provide strong integrity—it protects only against network attackers that intercept DNSSEC responses, but integrity is totally lost if the nameserver is compromised, because the nameserver holds the secret ZSK that can be used to sign bogus DNSSEC responses. We call this **weak integrity**.

RFC 7129 [42] describes an online signing approach called "NSEC3 White Lies" which is supported by at least one major nameserver implementation (PowerDNS). NSEC3 White Lies requires the nameserver to use the secret ZSK to generate, on the fly, an NSEC3 record that covers a query with the minimal pair of hash values.[8] That is, given a query α and its hash value $h(\alpha)$, the nameserver generates an NSEC3 record containing the pair of hashes $(h(\alpha)-1, h(\alpha)+1)$, and signs the NSEC3 record with the secret ZSK. Because the NSEC3 record only contains information about the queried name α, but not any name present in the zone, it provides **privacy against zone enumeration**. Offline zone enumeration attacks no longer work. Instead, enumeration is only possible by brute force—sending an online query to the nameserver for each name that is suspected to be in the zone. NSEC3 White Lies is also backwards-compatible for resolvers: resolvers just need to validate the NSEC3 record, but do not need to know or care whether the server is doing online signing (with NSEC3 White Lies) or not (with plain NSEC3).

[8] RFC4470 [87] also proposes "Minimally Covering NSEC Records" an analogous online signing approach that uses NSEC records instead of NSEC3 ones. We omit further discussion of this approach as it is not supported by major nameserver implementations (*i.e.,* BIND, PowerDNS, Microsoft DNS, *etc.*).

B SECURITY OF ECC-BASED VRF

We define the necessary security properties that a VRF needs to satisfy in order to be used in our application, and provide formal proofs that they are satisfied by ECC-based VRF from Figure 2.

It suffices to prove three properties: Trusted Uniqueness (see [65, Definition 10]), Selective Pseudorandomness (see [65, Definition 11]), and Collision-Resistance (not formally discussed in [65], but mentioned in the proof of Theorem 4). Sufficiency of these three properties for constructing NSEC5 follows from [65, Theorem 4]. We discuss each property in turn. We model the hash functions H_1 and H_3 as random oracles (we do not need to model H_2 as a random oracle—weaker assumptions suffice for H_2). We use notation $\mathrm{Ver}_{PK}(\alpha, \beta, \pi)$ to denote the verification algorithm, which outputs 1 if and only if the proof π and hash output β are valid for input α and public key PK.

Uniqueness. Recall that uniqueness requires that there should be only one provable VRF output β for every input α (this output is denoted by $F_{SK}(\alpha)$). *Trusted* uniqueness limits this requirement to only the case when the public key is valid. Following the VRF literature, Naor and Ziv [65, Definition 10]) define uniqueness unconditionally: for a validly generated public key, each input α to the VRF has at most one provably correct hash output β. However, the construction in Section 2.3 satisfies it only computationally: more than one hash output y may exist, but only the β produced by $F_{SK}(\alpha)$ can be proven correct by any computationally bounded adversary, even given the secret key. Since we are not aware of any prior work defining this relaxation of the uniqueness property (although Chase and Lysyanskaya [33] mention that such a relaxation can be defined), we define it here. Our definition is in terms of concrete, rather than asymptotic security, because this allows us to set length parameters.

Definition B.1. (Computational Trusted Uniqueness.) A VRF satisfies (Q_H, ϵ_U)-trusted uniqueness if for all adversaries A that make at most Q_H queries to the random oracle, for a key pair (PK, SK) produced according to the key generation algorithm, it holds that $\Pr[A(PK, SK) \to (\alpha, \beta_1, \pi_1)$ s.t. $\beta_1 \neq F_{SK}(\alpha)$ and $\mathrm{Ver}_{PK}(\alpha, \beta_1, \pi_1) = 1] \leq \epsilon_U$.

We now prove that the VRF satisfies Definition B.1 based on the randomness of the oracle H_3. Note that this proof does not rest on any computational assumptions or on programming a random oracle. Note also that it does not assume that H_1 is random (this assumption is needed for proofs of pseudorandomness and collision resistance, below), and would work essentially the same way even if the range of H_1 was the entire group G rather than $G - \{1\}$ (we exclude 1 to get a slightly cleaner proof of pseudorandomness).

Claim B.2. *The VRF satisfies (t_U, ϵ_U)-computational trusted uniqueness of Definition B.1 for $\epsilon_U = (Q_H + 1)/\min(q/2, \rho)$, where $\rho = |\mathrm{range}(H_3)|$ and $Q_H \leq t_U$ is the number of queries the adversary makes to the random oracle H_3.*

Note that the quantitative bound on ϵ_U in the above claim implies that the bit length $\log \rho$ of the output c of H_3 can be equal to the desired security parameter; in particular, it can be shorter than the prime order q of the group G (whose bit length needs to be at least twice the security parameter in order to protect against attacks on the discrete log). This claim is the only part of the security analysis affected by the output length of H_3 (and thus the bit length of the integer c from the VRF proof π).

Proof. Suppose there is an adversary A that violates computational trusted uniqueness with probability ϵ_U. That is, on input g, x, the adversary A makes Q_H queries to the H_3 oracle and wins by outputting (α, β_1, π_1) s.t. $\beta_1 \neq F_{SK}(\alpha)$ and $\mathrm{Ver}(\alpha, \beta_1, \pi_1) = 1$ with probability ϵ_U. We will show that $\epsilon_U \leq (Q_H + 1)/\min(q/2, \rho)$, where q is the order of the group G and $\rho = |\mathrm{range}(H_3)|$.

The proof π_1 contains γ_1 such that $\beta_1 = H_2(\gamma_1^f)$. Note that the correct $\beta = F_{SK}(\alpha)$ is computed as $H_2(\gamma^f)$ for $\gamma = [H_1(\alpha)]^x$. Since $\beta_1 \neq \beta$, we have $\gamma_1^f \neq \gamma^f$, i.e., $\gamma_1^f \neq h^{xf}$, where $h = H_1(\alpha)$. Now, it must be that $\pi_1 = (\gamma_1, c, s)$ for some c, s that ensure that $\mathrm{Ver}(\alpha, \beta_1, \pi_1) = 1$. The verification function Ver ensures that $\gamma_1 \in E$ and computes $h = H_1(\alpha)$ and $u = g^s PK^c$ and $v = h^s \gamma_1^c$. Because the VRF parameters and public keys are trusted, it follows that $g \in G$ and $PK = g^x \in G$. The range of H_1 is $G - \{1\}$ so $h \in G$. Since $G \subset E$, all variables in the above equations are in E.

For any $a \in E$, we define $\hat{a} = a^f$. By the structure theorem for finite abelian groups, E has exactly one subgroup of order q, because q does not divide f. This subgroup is $G = \{b \in E \mid b^q = 1\}$. Therefore, $\hat{a} \in G$, because $\hat{a}^q = a^{fq} = a^{|E|} = 1$ (by Fermat's little theorem).

We can now raise both equations to the power of the cofactor f to obtain similar equations, but with all the variables in G: $\hat{u} = \hat{g}^s \hat{PK}^c$ and $\hat{v} = \hat{h}^s \hat{\gamma}_1^c$. Note that $h \neq 1$ (since the range of H_1 is $G - \{1\}$). Because G is of prime order, h is also a generator of G. Since q does not divide f, $\hat{h} = h^f \neq 1$ and thus $\hat{h}$ is also a generator of G. Same for $\hat{g}$. Therefore we can take the logarithm of the first equation base $\hat{g}$ and that of the second one base $\hat{h}$. Solving these for s we get $\log_{\hat{g}} \hat{u} - cx \equiv s \pmod q$ and $\log_{\hat{h}} \hat{v} - c \log_{\hat{h}} \hat{\gamma}_1 \equiv s \pmod q$ which implies that

$$c \equiv \frac{\log_{\hat{g}} \hat{u} - \log_{\hat{h}} \hat{v}}{x - \log_{\hat{h}} \hat{\gamma}_1} \pmod q \tag{3}$$

Since $\hat{\gamma}_1 \neq \hat{h}^x$, the denominator is not zero, and so there is only one c modulo q that satisfies equation (3) given g, g^x, h, γ_1, u, and v. Recall that for verification to pass, it holds that $c = H_3(g, h, g^x, \gamma_1, u, v)$. Note that the contents of the query to H_3 contain every value in the right hand side of equation (3), and thus the correct c is uniquely defined at the time the query is made (assuming G is fixed).

What is the probability, for a given query to H_3, that the random value returned by the H_3 oracle is congruent to that correct c modulo q? Let ρ denote $|\mathrm{range}(H_3)|$. If the range of H_3 is a subset of $\{0, \ldots, q-1\}$, then this probability is either $1/\rho$ or 0, depending on whether the correct c is in $\mathrm{range}(H_3)$. Else (i.e., if $q < \rho$), think of reducing every element in $\mathrm{range}(H_3)$ modulo q. Then some values c modulo q will be hit $\lfloor \rho/q \rfloor$ times, while others will be hit $\lceil \rho/q \rceil$ times. Thus, the probability that any given c is hit is at most $\lceil \rho/q \rceil / \rho \leq ((\rho/q) + 1)/\rho = 1/q + 1/\rho < 2/q$. Assume the adversary outputs β_1, π_1 and then the verification algorithm is run. This causes a total of $Q_H + 1$ queries to H_3 (Q_H by A and one by the verifier), so by the union bound, the chances that any of them returns a correct c for that query are at most $(Q_H + 1)/\min(q/2, \rho)$. □

Remark. Our computational trusted uniqueness property is slightly weaker than the unconditional trusted uniqueness of Naor and Ziv's

[65, Definition 10]. Thus, the proof that NSEC5, when constructed from the VRF of Figure 2, satisfies the soundness property in [65, Theorem 4] needs a slight change, as follows. The proof in [65] is a reduction from an adversary A who violates soundness to an adversary B who forges signatures. The reduction relies on the fact that A must provide the correct β value (called y in [65]) and proof π for the VRF as part of its soundness-violating output on an input α (called x in [65]). Computational trusted soundness ensures that this happens except with negligible (i.e., $(Q_H+1)/\min(q/2,\rho)$) probability. Thus, the success probability of the reduction reduces from ϵ_U to $\epsilon_U - (Q_H+1)/\min(q/2,\rho)$.

Uniqueness without trusting the key. Our VRF can be modified to attain the stronger property of computational uniqueness (without needing to trust the key generation). There are three cases:

- If the group E is fixed and trusted to have been correctly generated (*i.e.,* E is known to have a subgroup of prime order q), and the generator g is known to be in $G-\{1\}$, then the verifier just needs to check that $PK \in E$. (This is the only requirement on PK from the proof above.)
- If the group E is fixed and trusted, but g and PK are not, then the verifier needs to check that $g \in E$, $g^f \neq 1$, as well as that $PK \in E$ (the scheme's design ensures that the expensive checks whether $g \in G$ and $PK \in G$ are not needed).
- If the group E is not fixed, then we need to include an unambiguous identifier of E as input to H_3 (so that a malicious prover cannot choose E after seeing c), and verifier needs to check that G is a subgroup of E of order q, q is prime, $|E| = qf$, q does not divide f, $g \in E$, $g^f \neq 1$, and $PK \in E$. Finally, the adversary should not be allowed to choose the mapping from E to its identifier after seeing c.

Pseudorandomness. We will state and prove pseudorandomness in terms of concrete, rather than asymptotic, security. This allows us to set parameters and work with fixed groups G, E.

In this section, we will need to use the notion of adversarial *advantage*. We remind the reader that it is defined as follows: the adversary can be used to play one of two games. At the end of each game, the adversary will output a single bit. The advantage is defined as the difference between the probabilities of outputting 1 in the two games.

The definition of pseudorandomness is as follows:

Definition B.3. (Pseudorandomness) A VRF satisfies $(t_P, Q_H, Q_P, \epsilon_P)$ pseudorandomness for output distribution S if no adversary D (which can depend on the fixed VRF parameters, such as G, E, etc.) whose running time and description size are bounded by t_P, total number of random oracle queries is bounded by Q_H and total number of Prove and F queries is bounded by Q_P, can distinguish the following two games with advantage more than ϵ_P. In both games, VRF keys (PK, SK) are honestly generated, and $D(PK)$ gets to query Prove_{SK}, F_{SK}, and the random oracles on arbitrary inputs. In both games, D chooses a challenge input α^* that has been queried to neither Prove nor F. In one game, D receives $F_{SK}(\alpha^*)$, while in the other D receives a random element drawn from S. Finally, after additional queries to Prove_{SK} and F_{SK} (except on α^*), D outputs one bit indicating which game D thinks it is playing. The weaker notion of $(t_{SP}, Q_H, Q_P, \epsilon_{SP})$ *selective* pseudorandomness is defined the same way, except D has to choose α^* before any queries and before seeing PK; D then gets either $F_{SK}(\alpha^*)$ or a random element from S as an input along with PK.

Note that our definition gives a slightly generalized version of the notions of pseudorandomness and selective pseudorandomness from [65, Definition 11]: instead of addressing only indistinguishability from the uniform distribution, it addresses indistinguishability from some specified distribution S. This allows us to present a modular proof of the pseudorandomness of our VRF.

For purposes of modularity, first we will not make any assumptions on H_2; rather, at first we will show that the output of our VRF is indistinguishable from H_2 applied to a uniformly random element of $G-\{1\}$, i.e., from the distribution $H_2(U_G)$, where U_G is the uniform distribution on $G-\{1\}$. We will then (in Claim B.6) address pseudorandomness for the uniform output distribution, by making an assumption on H_2.

Pseudorandomness of our VRF depends on the following assumption about the group G and generator g, known as the $(t_{DDH}, \epsilon_{DDH})$-*DDH (Decisional Diffie-Hellman) Assumption*: for any adversary C whose description size and running time are bounded by t_{DDH}, the difference in probabilities (where the probabilities are over a random choice of $h, h' \in G-\{1\}$ and $x \in \{1,\ldots,q-1\}$) that $C(g^x, h, h^x) = 1$ and $C(g^x, h, h') = 1$ is at most ϵ_{DDH}. (Because the assumption is specifically for the group G, we think of the fixed VRF parameters G, q, E, f, and g as hardwired into the adversary C.)

We note that pseudorandomness can also be proven from the Computational Diffie-Hellman (CDH) assumption if H_2 is modeled as a random oracle, but the security reduction will be lossier; we do not present this proof here.

We now prove that our VRF satisfies both pseudorandomness and selective pseudorandomness for output distribution $H_2(U_G)$. We address selective pseudorandomness first, because it is simpler. Our proof relies on programming the random oracles H_1 and H_3.

Claim B.4. *Under the $(t_{DDH}, \epsilon_{DDH})$-DDH assumption, for any Q_H, Q_P, the VRF satisfies $(t_{SP}, Q_H, Q_P, \epsilon_{SP})$ selective pseudorandomness for output distribution $H_2(U_G)$, for $t_{SP} \approx t_{DDH}$ (minus the time for $\Theta(Q_H+Q_P)$ exponentiations in G and one evaluation of H_2) and $\epsilon_{SP} = \epsilon_{DDH} + Q_P(Q_P+Q_H)/q$.*

Proof. We need to show the following: if

- D chooses α^*,
- then receives an honestly generated $PK = g^x$ and
  - either $H_2([H_1(\alpha^*)]^{xf})$
  - or H_2 applied to a random element of $G-\{1\}$,
- is allowed Q_H queries to random functions H_1 and H_3 and Q_P queries to Prove_{SK} or F_{SK} (except on α^*)
- can distinguish the two cases with advantage ϵ_{SP} after running time t_{SP}

then we can build C that breaks $(t_{DDH}, \epsilon_{DDH})$-DDH assumption for $t_{DDH} \approx t_{SP}$ (plus the time for $\Theta(Q_H+Q_P)$ exponentiations in G and one evaluation of H_2) and $\epsilon_{DDH} = \epsilon_{SP} - Q_P(Q_P+Q_H)/q$. Because F_{SK} is computable, in our case, from Prove_{SK} via Proof2Hash, we can assume without loss of generality that D never queries F_{SK}—every query to F_{SK} can be replaced with a query to Prove_{SK}.

Given (g^x, h, h') (where h' is either h^x or a random element of $G - \{1\}$), C gets α^* from D, sets the VRF public key PK as g^x and runs D with public key PK and input $H_2(h'^f)$. Note that if h' is a random element of $G - \{1\}$, then so is h'^f, because raising to the power f is a permutation of $G - \{1\}$, since q does not divide f. Thus, D is getting as input either the correct VRF output (because C will define $H_1(\alpha^*)$ to equal h) or $H_2(U_G)$, as required by Definition B.3.

C answers the queries of D as follows:

- If D queries α^* to random oracle H_1, C returns h.
- If D queries any other α_i to H_1, C first checks if this value has been queried to H_1 before; if so, C returns the same answer as previously returned. Else, C chooses a random $\rho_i \in \{1, \ldots, q-1\}$ and programs random oracle H_1 as $H_1(\alpha_i) := g^{\rho_i}$. (Note: this response is distributed uniformly in $G - \{1\}$, just like with the honest H_1, because g is a generator of G.)
- If D queries H_3, C first checks if this value has been queried to H_3 before; if so, C returns the same answer as previously returned. Else, C return a fresh random value in the appropriate range. (Note that these responses are distributed just like honest H_3).
- If D makes a query q_i to Prove_{SK} (note that $q_i \neq \alpha^*$),
  - C queries $H_1(q_i)$ as described above to get ρ_i,
  - C sets $\gamma = (g^x)^{\rho_i}$ where g^x was the public key given as input to D,
  - C chooses random values $s \in \{0, \ldots, q-1\}$ and $c \in \text{range}(H_3)$ and then computes $u = g^s(g^x)^c$ and $v = [g^{\rho_i}]^s[(g^x)^{\rho_i}]^c$. (Note that $u, v, x, h = g^{\rho_i}, s$, and c are distributed identically to the distribution produced by Prove. The difference in how these distributions are obtained is simply that Prove chooses a uniform k while C chooses a uniform s, where k and s are tied by the equation $s + cx \equiv k \pmod q$, and $u = g^k, v = h^k$.) If $H_3(g, g^{\rho_i}, g^x, (g^x)^{\rho_i}, u, v)$ is already defined, then C fails and aborts. Else, C programs the random oracle H_3 to let $H_3(g, g^{\rho_i}, g^x, (g^x)^{\rho_i}, u, v) := c$. (Note: if C does not abort, then H_3 is uniformly random, just like honest H_2 and H_3).

If C does not abort, then its simulation for D is faithful and C can just output what D outputs. The probability that C aborts is simply the probability that $H_3(g, g^{\rho_i}, g^x, (g^x)^{\rho_i}, u, v)$ is already defined during the computation of the response to Prove; since at most $Q_H + Q_P$ values of H_3 are defined, and u is a uniformly random value in G (because s is uniformly random in $\{0, \ldots, q-1\}$ and g is a generator), the chances that a single query to Prove causes an abort are $(Q_H + Q_P)/q$, and the chances that any of the queries to Prove causes an abort are $Q_P(Q_H + Q_P)/q$. Thus, the advantage of C is at least $\epsilon_{SP} - Q_P(Q_P + Q_H)/q$. □

We can also prove pseudorandomness, but with a lossier security reduction than selective pseudorandomness.

Claim B.5. *Under the $(t_{DDH}, \epsilon_{DDH})$-DDH assumption, for any $Q_H \geq 1, Q_P$, the VRF satisfies $(t_P, Q_H, Q_P, \epsilon_P)$ pseudorandomness for output distribution $H_2(U_G)$, for $t_P \approx t_{DDH}$ (minus the time for $\Theta(Q_H + Q_P)$ exponentiations in G and an evaluation of H_2) and $\epsilon_P = 4\epsilon_{DDH} \cdot Q_P + Q_P(Q_P + Q_H)/q$.*

Proof. We explain the proof by showing the differences from the proof of Claim B.4. The problem is that C does not know what α^* is—it could be in any of the H_1 queries. We follow the approach of [36] to deal with this problem.

Whenever D makes a query α_i to H_1, C flips a biased coin to decide whether this query is going to be "type-sig" (with probability $Q_P/(Q_P + 1)$) or "type-attack" (with probability $1/(Q_P + 1)$). If the query is "type-sig," then C works the same way as in the proof of Claim B.4. Else, C returns h^{ρ_i} for a random $\rho_i \in \{1, \ldots, q-1\}$. C remembers the type of the query and the ρ_i value. If D makes a query q_i to Prove, then C aborts if $q_i = \alpha_i$ for an α_i of type-attack (else C proceeds as before). At some point D produces α^*; before proceeding, C makes sure α^* has been queried to H_1 (performing the query if it hasn't been). C aborts if $\alpha^* = \alpha_i$ for some α_i of type-sig, and otherwise returns $H_2(h'^{\rho_i f})$ as the response to the challenge.

We note that all the responses to H_1 queries are still uniformly distributed over $G - \{1\}$ and independent, because both g and h are generators of G. If $h' = h^x$, then D receives the correct value for $F_{SK}(\alpha^*)$, namely $H_2(h'^{\rho_i f}) = H_2(h^{x\rho_i f}) = H_2([H_1(\alpha^*)]^{xf})$. On the other hand, if h' is a uniform element of $G - \{1\}$, then instead of $F_{SK}(\alpha^*)$, D receives a uniform response chosen independently of anything else from $H_2(G - \{1\})$, because a uniform value in $G - \{1\}$ raised to f (a fixed power not divisible by q) is uniform in $G - \{1\}$.

Now C succeeds as long as (1) there is no abort due to a collision of H_3 inputs as in the proof of Claim B.4) and (2) the guesses for the H_1 query type (type-sig or type-attack) don't lead to an abort. Note that these guesses are independent of the view of D and therefore of the success of D. The probability that the guesses are correct for each Prove query and for α^* is $\left(\frac{Q_P}{Q_P+1}\right)^{Q_P} \frac{1}{Q_P+1} \geq \frac{1}{4Q_P}$ whenever $Q_P \geq 1$. (The bound is obtained by observing that the left-hand side multiplied by Q_P is increasing for $Q_P \geq 1$, and its value at $Q_P = 1$ is 1/4.) We thus obtain the claimed result. □

Proving pseudorandomness for uniform output distribution requires making an assumption on H_2. Namely, we assume that $H_2(U_G)$ is indistinguishable from the uniform distribution on bit strings of length 2ℓ, where 2ℓ is output length of H_2. The assumption, known as $(t_{PH_2}, \epsilon_{PH_2})$-*pseudorandomness of* H_2, is as follows: for any adversary C whose description size and running time are bounded by t_{PH_2}, the difference in probabilities (where the probabilities are over a random choice of $\gamma \in G - \{1\}$ and $\beta \in \{0,1\}^{2\ell}$) that $C(H_2(\gamma)) = 1$ and $C(\beta) = 1$ is at most ϵ_{PH_2}. This assumption is strictly weaker than modeling H_2 as a random oracle.

Claim B.6. *Under the $(t_{DDH}, \epsilon_{DDH})$-DDH assumption and $(t_{PH_2}, \epsilon_{PH_2})$-uniformity of the H_2 assumption, for any Q_H, Q_P, the VRF satisfies $(t_{SP}, Q_H, Q_P, \epsilon_{SP})$ selective pseudorandomness for output distribution U_{H_2}, for $t_{SP} \approx \min(t_{DDH}, t_{SP})$ (minus the time for $\Theta(Q_H + Q_P)$ exponentiations in G and one evaluation of H_2) and $\epsilon_{SP} = \epsilon_{DDH} + Q_P(Q_P + Q_H)/q + \epsilon_{PH_2}$. The VRF also satisfies $(t_P, Q_H, Q_P, \epsilon_P)$ pseudorandomness for output distribution U_{H_2}, for t_P defined the same way as t_{SP} and $\epsilon_P = 4\epsilon_{DDH} \cdot Q_P + Q_P(Q_P + Q_H)/q + \epsilon_{PH_2}$.*

Proof. We prove the first part of the claim. The proof is by a standard hybrid argument. Suppose an adversary D breaks pseudorandomness for output distribution U_{H_2} with advantage more than

ϵ_{SP}. We know, by Claim B.4, that D cannot distinguish whether its second input is equal to $F_{SK}(\alpha^*)$ or chosen from $H_2(U_G)$ with advantage more than $\epsilon_{DDH} + Q_P(Q_P + Q_H)/q$. Therefore, D must distinguish whether its second input is chosen from $H_2(U_G)$ or U_{H_2} with advantage at least $\epsilon_{SP} - (\epsilon_{DDH} + Q_P(Q_P + Q_H)/q) = \epsilon_{PH_2}$, by the triangle inequality. But when D's second input is $H_2(U_G)$ or U_{H_2}, all the other information D receives can be faithfully simulated. Thus, D can be used to build a distinguisher C to violate the uniformity of H_2: specifically, on input b (which is either $H_2(\gamma)$ or β), C will generate a VRF key pair, run D on input (PK, b), answer the queries of D by running the VRF algorithms, and output whatever D outputs.

The proof of the second part is essentially identical, except it uses Claim B.5 instead of Claim B.4, and the reduction C provides the value b not as input to D, but as a response to the adaptively chosen challenge query α^*. □

Collision Resistance. We now define trusted collision resistance, which states that an adversary cannot produce a collision even given SK, as long as the keys are honestly generated. This property, while not explicitly defined in [65], is necessary to ensure the completeness of NSEC5, i.e., to ensure that a valid non-existence proof can always be generated by the nameserver and accepted by the resolver whenever the record does not exist (see [65, Proof of Theorem 4]).

Definition B.7. (Trusted Collision Resistance) A VRF satisfies (Q_H, ϵ_C) trusted collision resistance if no adversary making Q_H random oracle queries, can, given an honestly generated SK, output two values $\alpha \neq \alpha'$ such that $F_{SK}(\alpha) = F_{SK}(\alpha')$ with probability greater than ϵ_C.

Proving collision resistance requires a mild assumption about the structure of H_2: namely, that H_2 of two random inputs is unlikely to produce the same output. This assumption is strictly weaker than modeling H_2 as a random oracle.

Claim B.8. *Assume that the probability that two uniformly random elements γ_1, γ_2 of $G - \{1\}$ satisfy $H_2(\gamma_1) = H_2(\gamma_2)$ is at most ϵ_{CH_2}. Then our VRF satisfies ϵ_C-trusted collision resistance for $\epsilon_C = \epsilon_{CH_2} \cdot (Q_H + 2)^2/2$.*

Proof. Let α, α' be the output of the adversary. Without loss of generality, assume α and α' have been queried to H_1; if not, add those queries to the code of the adversary, for a total of $Q_H + 2$ queries.

Recall that in the random oracle model, H_1 is assumed to be chosen uniformly after the adversary is designed; in particular, H_1 should be different for each public key PK (this can be accomplished by using PK as a hash function salt). Let $\alpha_1, \ldots, \alpha_{Q_H+2}$ be the queries the adversary makes to H_1. Given two values $\alpha_i \neq \alpha_j$, what is the probability (for a random choice of the oracle H_1) that $F_{SK}(\alpha_i) = F_{SK}(\alpha_j)$? Such a collision happens if $[H_1(\alpha_i)]^{xf}$ collides with $[H_1(\alpha_j)]^{xf}$ after the application of H_2. Since $H_1(\alpha_i)$ and $H_1(\alpha_j)$ are uniform in $G - \{1\}$ and raising to the power xf is a permutation of $G - \{1\}$, the probability they collide is at most ϵ_{CH_2}. Applying the union bound over at most $(Q_H + 2)^2/2$ pairs of distinct queries to H_1, we get that a successful output α, α' exists among queries to H_1 with probability at most $\epsilon_{CH_2}(Q_H + 2)^2/2$. □

Collision resistance without trusting the key. Similarly to the case of uniqueness, our VRF can be modified to attain collision resistance without needing to trust the key generation. The modifications are the same as in the case of uniqueness (to ensure that F_{SK} is uniquely defined), with the additional check that $PK^f \neq 1$ to ensure that x is not divisible by q.

B.1 Hashing onto the Curve

The ECC-based VRF (Figure 2) uses a hash function H_1 that maps arbitrary-length strings to points on an elliptic curve. How can we instantiate such a hash function? Ideally we want an instantiation that works for both curves we have considered: NIST P-256 and Ed25519.

One lightweight technique was proposed in [29]. It proceeds as follows. Assume an elliptic curve with equation $y^2 = x^3 + ax + b$ and order qf. Given an input α (the queried name in our case), set counter $i = 0$ and compute $h = H(PK, \alpha, i)$, where H is a standard cryptographic hash function, *e.g.,* SHA-256. Then, if $h^3 + ah + b$ is a quadratic residue (that is, h is the valid x-coordinate of a point on the curve) output the point $(h, (h^3 + ah + b)^{1/2})$ raised to the power of cofactor f (there are generally two square roots; which one to choose can be determined by having one additional output bit of H, or the numerically smaller of the two roots can be chosen deterministically, which will cut the hash range in half and will not significantly affect security). Otherwise, increment the counter by 1 and try again. This simple process is expected to terminate after two steps, and the involved operations are very fast, with an expected running time of $(O \log^3(n))$, if the curve is defined over finite field $GF(n)$. The range of this function is only half of the group G (if only one y is chosen for a random x), but that does not materially change the proofs of security.

As first shown in [31], the above technique is not suitable when α must be kept secret; this is because the running time of the hashing algorithm depends on α, and so it is susceptible to timing attacks. However, this attack is not relevant for NSEC5, because the only value hashed in the query phase is the query α itself, which is already known to the adversary.

Other options for instantiating H_1 are discussed in [78]. Many of these options do not produce a uniform element of $G - \{1\}$. Rather, they use a hash function (which we will model as a random oracle) to get a uniform bit string in some range $\{0, 1\}^t$ for $2^t > |G|$ and then map the bit string to E using a map M that covers a large portion (e.g., a quarter) of E and is such that every element of G has not too many (e.g., at most 8) preimages. The resulting value in E is then raised to the cofactor f to obtain a value in G. We need to show that our proofs still hold for such instantiation of H_1.

B.2 Security of Hashing onto the Curve

Specifically, assume H_1 is instantiated as follows. First map input α to a uniform bit string in the range $\{0, 1\}^t$, via a random oracle J. Then map $J(\alpha)$ to an element of E via a map M. Then map the element of E to an element of G by raising to the cofactor f. Thus, $H_1(\alpha) = M(J(\alpha))^f$. Assume M has the following properties: any element of E has at most d preimages under M (i.e., M is at-most-d-to-1) and these preimages can efficiently computed (i.e., M is

invertible). We will now describe the changes to the security proof that are necessary to accommodate such H_1.

First, let us introduce the following algorithm **preimage-sample**(b) to sample a uniform value j such that the discrete logarithm of $M(j)^f$ is known. On input $b \in G - \{1\}$, it returns a uniformly random $j \in \{0,1\}^t$ and $z \in [0, q-1]$ such that $M(j)^f = b^z$.

(1) Let T be the f-torsion subgroup of E, i.e., the set of all elements τ such that $\tau^f = 1$. Note that it is easy to obtain a uniform element of T by taking a uniform element of E and raising it to the power q.
(2) *done* = false
(3) repeat until *done*
    - pick uniformly at random $y \in 0, \ldots, q-1$
    - pick uniformly at random $\tau \in T$
    - With probability $|M^{-1}(\tau b^y)|/d$, set *done* = true
(4) Sample j uniformly from $M^{-1}(\tau b^y)$
(5) Output j and $z = yf \bmod q$

Observe that the output is correct: $M(j)^f = (\tau b^y)^f = b^{yf} = b^z$. We will now prove that this algorithm outputs a uniform j and bound the algorithm's expected running time and the probability $z = 0$. Let $\delta = dqf/2^t$. Note that if $2^t \geq qf = |E|$, then $\delta \leq d$.

Claim B.9. *The above procedure* ***preimage-sample*** *returns a uniform j. the expected number of iterations before the output is produced is δ.* $\Pr[z = 0] \leq df/2^t$.

Proof. To show that j is uniform, if suffices to show that every j is equally likely to be output by a given iteration. Note that τb^y is uniform in E (because b is a generator of G, so b^y is uniform in G; τ is uniform in T; and E is isomorphic to $T \times G$ by the structure theorem for finite abelian groups). Fix $j \in \{0,1\}^t$; let $s = |M^{-1}(M(j))|$. $\Pr_z[j \text{ is output in a given round}] = \Pr_z[\tau b^y = M(j)] \cdot \Pr[\textit{done} \text{ is set to true}] \cdot \Pr[j \text{ is picked from } M^{-1}(\tau b^y)] = (1/(|E|)) \cdot (s/d) \cdot (1/s) = 1/(|E| \cdot d) = 1/(dqf)$, regardless of j.

To show that the expected number of iterations is δ, we need to show that each iteration sets *done* to true with probability $1/\delta$. It is easier to see this if we think of line 4 as being inside the loop (this change does not change the number of iterations or the output of the algorithm). Then each iteration outputs a given j with probability $1/(dqf)$, as proven in the previous paragraph. There are 2^t possible values of j than an iteration could output. Thus, each iteration succeeds with probability $2^t/(dqf) = 1/\delta$.

Finally, that because j is uniform, $\Pr[z = 0] = \Pr_{j \in \{0,1\}^t}[M(j)^f = 1] = \Pr_{j \in \{0,1\}^t}[M(j) \in T] \leq d|T|/2^t = df/2^t$, because T has f elements and each of them has at most d preimages under M. □

We now investigate the three properties (uniqueness, pseudorandomness, and collision-resistance) when $H_1(\cdot)$ is replaced by $M(J(\cdot))$.

The proof of uniqueness does not change.

For the proof of selective pseudorandomness, let us introduce $(M, t_{MDDH}, \epsilon_{MDDH})$-*DDH Assumption*: for any adversary C whose description size and running time are bounded by t_{DDH}, the difference in probabilities (where the probabilities are over a uniform choice of $j \in \{0,1\}^t$ such that $M(j)^f \neq 1$, a uniform choice of $h' \in G - \{1\}$, and a uniform choice of $x \in \{1, \ldots, q-1\}$) that $C(g^x, j, M(j)^{xf}) = 1$ and $C(g^x, j, h') = 1$ is at most ϵ_{DDH}.

Claim B.10. *If $(t_{DDH}, \epsilon_{DDH})$-DDH assumption holds, then the $(M, t_{MDDH}, \epsilon_{MDDH})$-DDH assumption holds for $\epsilon_{MDDH} = \epsilon_{DDH} + df/2^t$ and $t_{MDDH} = t_{DDH}$− running time of* ***preimage-sample****.*

Proof. Suppose an adversary C whose description size and running time are bounded by t_{MDDH} is able to distinguish $(g^x, j, M(j)^{xf})$ from (g^x, j, h') with advantage more than ϵ_{MDDH}. We will build C' as follows: given g^x, h, h' as input, the goal of C' is to distinguish the case of $h' = h^x$ from the case of h' being uniform in $G - \{1\}$. To do so, C' will run **preimage-sample**(h) to get j and z. By Claim B.9 above, j is uniform in $\{0,1\}^t$. If $z = 0$ (i.e., $M(j)^f = 1$), C' will output 0. Else C' will run C on g^x, j, h'^z. When $h' = h^x$, then $h'^z = h^{xz} = M(j)^{xf}$. When h' is uniform in $G - \{1\}$, then h'^z is uniform in $G - \{1\}$ for any non-zero 0 (because raising to the power z is a permutation of $G - \{1\}$). Thus, C will distinguish the two cases with advantage $\epsilon_{MDDH} - \Pr[z = 0]$. □

Switching from $H_1(\cdot)$ to $M(J(\cdot))$ will change the statements of Claims B.4, B.5, and B.6 as follows: the values of t_P and t_{SP} will get reduced not by $\Theta(Q_H + Q_P)$ exponentiations in G but by $\Theta(\delta(Q_H + Q_P + 1))$ exponentiations in G and evaluations of M^{-1}; the values of ϵ_p and ϵ_{SP} will get increased by $df(Q_H + Q_P + 1)/2^t$.

The easiest way to prove these claims is to first rely on the $(M, t_{MDDH}, \epsilon_{MDDH})$-DDH assumption, and subsequently convert to the $(t_{DDH}, \epsilon_{DDH})$-DDH assumption per Claim B.10, losing δ exponentiations in G and evaluations of M^{-1} in running time and $df/2^t$ in adversarial advantage.

The proof of Claim B.4 will then require the following changes. First, because of $(M, t_{MDDH}, \epsilon_{MDDH})$-DDH assumption, C is given (g^x, j, h') rather than (g^x, h, h'). Second, D will query J rather than H_1. The value j is how C will respond to D's J-query on α^*. For queries to any other α_i, C will run **preimage-sample**(g) to get j_i and ρ_i such that $M(j_i)^f = g^{\rho_i}$; C will abort if $\rho_i = 0$; else, C will return j_i to D.

The proof of Claim B.5 will require similar changes. Additionally, C will set $h = M(j)$. If a J-query is "type-attack", C will run **preimage-sample**(h) to get ρ_i and j_i such that $M(j_i) = h^{\rho_i}$; C will abort if $\rho_i = 0$; else, C will return j_i to D.

The proof of Claim B.6 will remain the same.

The proof of collision-resistance now requires a different assumption on H_2: namely, that $H_2(M(\cdot))$ on two random inputs is unlikely to produce the same output. If d is small, then this assumption is still strictly weaker than the assumption that H_2 is a random oracle.