Making NSEC5 Practical for DNSSEC

Dimitrios Papadopoulos*
HKUST & University of Maryland

Duane Wessels
Verisign Labs

Shumon Huque*
Salesforce

Moni Naor
Weizmann Institute

Jan Včelák†
NS1

Leonid Rezyin
Boston University

Sharon Goldberg
Boston University

ABSTRACT

NSEC5 is a new proposal for providing *authenticated denial of existence* for DNSSEC, *i.e.,* for providing authenticated responses to DNS queries for names that do not exist in a zone. NSEC5 simultaneously 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. By contrast, today's DNSSEC protocol can guarantee one of these properties, but not both. This paper argues that NSEC5 can be made practical and performant.

To that end, we present a new version of NSEC5. Our NSEC5 redesign features a fast and efficient verifiable random function (VRF) based on elliptic curve cryptography, along with a new cryptographic proof of its security. We also redesign the DNS protocol surrounding NSEC5, leveraging precomputation to improve performance and DNS-level optimizations to shorten responses. Next, we present the first implementation of NSEC5—extending widely-used DNS software to present a full-fledged nameserver and recursive resolver that support NSEC5—and evaluate their performance under aggressive query loads. We believe that our performance results indicate that NSEC5 can be a practical solution for DNSSEC deployments.

KEYWORDS

DNSSEC, verifiable random functions, elliptic curve cryptography, implementation

1 INTRODUCTION

The Domain Name Security Extensions (DNSSEC) uses asymmetric cryptography to protect the integrity and authenticity of DNS responses. NSEC5 [48] is a new proposal for providing *authenticated denial of existence* for DNSSEC, *i.e.,* for responding 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 security 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). Hardening the DNS against external compromise seems to be an increasingly important security goal [71], especially in light of recent attacks [1, 2, 4, 39, 53].

Second, NSEC5 provides *privacy* against offline *zone enumeration* [16, 24, 28, 60, 67, 70, 84, 85, 87], where an adversary makes a small number of online DNS queries and then processes them offline in order to learn all the domain names in a zone. Zone enumeration can be used to identify routers, servers or other 'things' (thermostats, fridges, baby monitors, *etc.*) that could then be targeted in more complex attacks. An enumerated zone can also 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" [60] (*e.g.,* per EU data protection laws [75],[19, pg. 37]). Several publicly available network reconnaissance tools can be used to launch zone-enumeration attacks [10, 16, 28, 67, 70, 84].

While today's DNSSEC protocol has several mechanisms for authenticated denial of existence, they all either fail to provide integrity against a compromised nameserver (*i.e.,* online signing used in NSEC3 White Lies [44] and Minimally-Covering NSEC [86]), or fail to prevent offline zone enumeration (NSEC [20], NSEC3 [60]). In fact, offline zone enumeration is an issue introduced by DNSSEC, and is not a possible attack on legacy DNS.

The original NSEC5. NSEC5 was first proposed in [48]. This first proposal, which lacked a full specification and implementation, was met with some skepticism [47, 82].

The first issue is that when DNSSEC uses schemes that do *not* prevent offline zone enumeration, then DNSSEC responses can be precomputed. By contrast, NSEC5 requires an online asymmetric cryptographic computation at the nameserver, in response to every negative DNSSEC query. (This is *necessary.* As shown in [48], online cryptography is necessary for *any* scheme that both (a) provides integrity, and (b) prevents zone enumeration.) Thus, there was a concern that NSEC5 would not be sufficiently performant.

The second issue is the length of 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 packet [69, 70, 72]. Long responses sent over UDP can be fragmented across multiple IP fragments, and thus risk being dropped by a middlebox that blocks IP

*Substantial parts of this work conducted at Verisign Labs.
†Substantial parts of this work conducted at CZ.NIC..

fragments [76, 80] or being subject to an IP fragmentation attack [50]. Alternatively, the resolver can resend the query over TCP [37, 65], harming performance (due to roundtrips needed to establish a TCP connection) and availability (because some middleboxes block DNS over TCP) [76]. Worse yet, long DNSSEC responses can be used to amplify DDoS attacks [43]. 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.

The NSEC5 proposal in [48] was based on RSA, which exacerbated both concerns, because of the length of an RSA modulus and the cost of an RSA exponentiation. Under the proposal, each NSEC5 response would contain up to three additional (long) RSA values that had to be computed on-the-fly. Moreover, there is currently serious discussion about replacing RSA, which is widely used in DNSSEC deployments [18, 78], with elliptic curve cryptography (ECC) [52, 77, 81]; the goal is to have shorter responses at a better security level. Thus, there was little enthusiasm for a new scheme based on RSA.

In this paper, we implement and evaluate the NSEC5 proposal from [48], and find that the concerns about its performance and response lengths were justified.

A new version of NSEC5. In order to support the security goals of NSEC5 without incurring the costs of the original RSA-based NSEC5 proposal, we set out to design a new version of NSEC5. Our approach proceeds along two lines.

First, we introduce DNS-level optimizations (Section 5) that allow us to (1) precompute parts of the response, and (2) reduce the number of DNSSEC records in the response.

Second, we redesign the cryptography behind NSEC5 (Section 4), introducing a scheme based on elliptic curve cryptography (ECC). To maintain the security properties of NSEC5, we cannot just replace RSA with ECDSA. (Why? See Section 4.2.) Instead, the starting point for our work is the observation of [66] that NSEC5 can be generically constructed from a verifiable random function (VRF) [63]. A VRF is the public-key version of a keyed cryptographic hash. We construct a VRF based on ECC, and prove its security in the random oracle model. While our ECC VRF is similar to a construction implicit in [41], this earlier work both lacked a proof of security, and failed to satisfy the VRF security properties due to a critical design flaw (that has been corrected as a result of our work [7, 8, 40]). Beyond this, we take special care to minimize the length of our VRF's outputs while still maintaining security. Our VRF has been submitted for standardization at the IETF [49].

Implementation. Our new version of NSEC5 has been submitted for standardization at the IETF [83]. To evaluate our new version of NSEC5, we present a full implementation of an authoritative nameserver and recursive resolver that support both RSA- and ECC-based NSEC5 (Section 7). (For the nameserver implementation, we extend the Knot DNS 1.6 [12]. For the recursive resolver, we extend Unbound 1.5.9.)

Performance results. Even though NSEC5 necessarily requires the nameserver to perform online cryptographic computations, we find that our new ECC-based NSEC5 can be viable even for high-throughput scenarios. Throughput at our authoritative nameserver easily scales to a few tens of thousands of queries per second (64K query/second) on a moderately-sized multi-core server (*i.e.,* 24 threads on 40 virtual cores). This is an order of magnitude larger than the average negative response rate at single server in the DNS's root zone [6]. In fact, our ECC-based NSEC5 nameserver implementation achieves a throughput that is about 2x higher than the only nameserver implementation that prevents offline zone enumeration, is widely deployed, and is compliant with the DNSSEC standards (*i.e.,* PowerDNS's implementation of online signing via NSEC3 White Lies [17]). Also, the performance of our NSEC5-ready recursive resolver is comparable to DNSSEC's existing denial-of-existence mechanisms.

Response lengths. We show (Section 8.1) that our ECC-based NSEC5 responses fit into a single IP packet, 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 [18, 78]), which has a lower security level!

Considering the transition to NSEC5. We conclude (Section 11) by discussing mechanisms for transitioning NSEC5 into the DNSSEC protocol. Given that the adoption of new cryptographic algorithms into DNSSEC may be on the horizon (*e.g.,* digital signatures over Edwards elliptic curves [77, 88]), now may also be a good time to consider the transition to NSEC5.

Contributions. We make the following contributions:

- We present a VRF based on elliptic curves, prove its security in the random oracle model, and use it to design a more performant version of NSEC5 (Section 4, Appendix B).
- We design the DNS protocol surrounding NSEC5, using precomputation and other optimizations to improve performance and shorten response lengths (Section 5).
- We present the first full-fledged implementation of both RSA- and ECC-based NSEC5 for both an authoritative nameserver and a recursive resolver. Our evaluation highlights significant improvements in throughput and response size achieved by our new ECC-based NSEC5 (Section 7,8).
- We discuss challenges and opportunities for adopting NSEC5 in practice (Section 11).

2 TRADEOFFS IN TODAY'S DNSSEC

We start by reviewing the issues that lead to the development of NSEC5 for DNSSEC. (See *e.g.,* [87] for a historical overview of the full DNSSEC protocol.) 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. We describe each scheme and its tradeoffs below:

NSEC (RFC 4034 [20]). 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 private *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 intercept and 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 private ZSK in order to produce a valid NSEC record. Without the private 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 [10, 14, 67, 70].

NSEC3 (RFC 5155 [60]). 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 private 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 the associated DNSSEC signatures) for the pair of hashes lexicographically before and after the *hash* of the non-existent name.

When NSEC3 records are precomputed, it also provides strong integrity. However, [28, 85] demonstrated (and RFC 5155 [60, 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 [14, 16, 84] and technologies for fast hashing continue to improve (*e.g.,* GPUs [85], ASICs).

Online signing with NSEC3 White Lies (RFC 7129 [44]). 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**.

	no online crypto	weak integrity	strong integrity	privacy
legacy DNS	✓	*X*	*X*	✓
(plain) NSEC or (plain) NSEC3	✓	✓	✓	*X*
online signing, e.g. NSEC3 White Lies	*X*	✓	*X*	✓
NSEC5	*X*	✓	✓	✓

Table 1: Properties of NSEC*. Note that [48] proved that it is impossible to provide both privacy and weak integrity without online crypto.

RFC 7129 [44] 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.[1] 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 private 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 also has a helpful backwards-compatiblity property 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).

3 SECURITY PROPERTIES OF NSEC5

NSEC5 was introduced in [48, 66], to provide both privacy against zone enumeration and strong integrity. NSEC5 is very similar to NSEC3, except that we replace the cryptographic hashes used in NSEC3 with the hashes computed by a *verifiable random function (VRF)* [63]. Table 1 summarizes properties of NSEC5. We now review the security properties of NSEC5, and revisit the exposition in [66] to show how NSEC5 can be generically constructed from a VRF.

3.1 Verifiable Random Functions (VRF).

A VRF [63] is essentially the public-key version of a keyed cryptographic hash. A VRF comes with a public-key pair (PK, SK). Only the holder of the private key SK can compute the hash, but anyone with public key PK can verify the hash.

[1] RFC4470 [86] also proposes "Minimally Covering NSEC Records" an analogous online signing approach that uses NSEC records instead of NSEC3 records. We omit further discussion of this approach because it is not supported by major nameserver implementations (*i.e.,* BIND, PowerDNS, Microsoft DNS, Knot DNS, *etc.*).

A VRF hashes an input α using the private key SK

$$\beta = F_{SK}(\alpha)\,.$$

The **collision-resistance** guarantee of a VRF is similar to that of a cryptographic hash function. The **pseudorandomness** of a VRF guarantees that β is indistinguishable from random by anyone who does not know the private key SK. The private key SK is also used to construct a *proof* π that β is the correct hash output

$$\pi = \Pi_{SK}(\alpha)\,.$$

The proof π is constructed in such a way that anyone holding the public key can validate that indeed $\beta = F_{SK}(\alpha)$. Finally, the VRF has a **trusted uniqueness** property that roughly requires that, given the VRF public key PK, each VRF input α corresponds to a unique VRF hash output β. More precisely, trusted uniqueness guarantees that, given a validly-generated PK, even an adversary that knows SK cannot produce a valid proof for a fake VRF hash output $\beta' \neq \beta$. (The word "trusted" here is used to indicate that we trust the key generation process, and are not concerned with uniqueness for untrusted keys.) See Appendix B for formal definitions.

All the VRFs we consider in this paper allow β to be computed directly from π by a simple operation, *i.e.,* hashing. This reduces communication, since communicating π alone (without β) suffices.

3.2 NSEC5 from VRFs.

NSEC5 uses a VRF to provide authenticated denial of existence for DNSSEC [66, Sec. 7]. We review the NSEC5 construction and three new types of DNSSEC records it requires: 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 computes DNSSEC signatures. The private VRF key is known to both the nameserver and the trusted owner of the zone. Meanwhile, the private 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 standard mechanisms used for DNSSEC key distribution.

Why do we need two separate keys, namely the ZSK (for signing DNS records) and the VRF key (for NSEC5)? This allows us to separate our two security goals (*i.e.,* strong integrity and privacy against zone enumeration). To achieve strong integrity, we follow the approach in NSEC and NSEC3, and provide the private ZSK to the the trusted zone owner but not to the untrusted nameserver. On the other hand, 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 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 [48], cryptographic lower bounds were used to prove the nameserver must *necessarily* have some secret cryptographic key. However, we shall soon see that NSEC5 still provides strong integrity even if the nameserver's private key is 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 (Table 2).

	integrity	privacy
Online signing	X	X
NSEC5	✓	X

Table 2: Comparing online signing (*e.g.,* NSEC3 White Lies) to NSEC5 when the nameserver is compromised.

Precomputing NSEC5 records. The trusted zone owner uses the private VRF key SK to compute the VRF hashes of all the names present in the zone, lexicographically orders all the the hash values, and uses the private ZSK to sign a record containing every consecutive pair of hashes; each pair of hashes is an NSEC5 record. The precomputed NSEC5 records and their associated DNSSEC signatures are provided to the nameserver along with the private VRF key SK.

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

(1) an NSEC5PROOF record containing π, and[2]
(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 exactly why NSEC3 is vulnerable to offline zone enumeration: because its hash function is publicly computable!)

Validating. The resolver validates the response by

(1) using the public VRF key in the NSEC5KEY record to validate that proof π from the NSEC5PROOF corresponds to the query α,
(2) using a simple operation (*i.e.,* hashing) to get β from π and then checking that β falls between the two hash values in the NSEC5 record, and
(3) using the public ZSK to validate the DNSSEC signatures on the NSEC5 record.

3.3 Properties of NSEC5.

Table 1 summarizes the properties of NSEC5.

Online crypto. NSEC5 requires online cryptographic computations for negative responses. (But not for positive responses.) For every query α that elicits a negative response, the nameserver uses the secret VRF key SK to compute the

[2]We use VRFs where β can be publicly computable from the proof π, so do not include β in the NSEC5PROOF record. VRFs that do not have this property additionally require β to be included in the NSEC5PROOF.

NSEC5PROOF record on the fly. Notice that online signing (*e.g.,* 'NSEC3 White Lies', see Section 2) also requires online cryptographic computations. The fact that both of these solutions prevent zone enumeration is not a coincidence: [48] proved that any solution that both (a) prevents zone enumeration and (b) provides weak integrity, must *necessarily* use online cryptography. What is interesting about NSEC5 is that it provides strong integrity (*i.e.,* integrity even when the nameserver is malicious or compromised). Meanwhile, online signing provides only weak integrity (*i.e.,* against network attackers but not compromised nameservers). See Tables 1-2.

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.

To see why, 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 private VRF key, 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 [66].

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 because the untrusted nameserver does not know the secret zone-signing key (ZSK). The idea behind the formal proof (see [66]) 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.[3] 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 private NSEC5KEY is leaked to an adversary, the security of NSEC5 just downgrades to that of (plain) NSEC3. (See Tables 1-2.)

4 REDESIGNING THE CRYPTO

As discussed in Section 1, a key problem with the original NSEC5 construction from [48] was that it was based on RSA. We first review [48]'s NSEC5 construction and explain why it implicitly contains an RSA-based VRF; we prove the security of this RSA-based VRF in Appendix C. We then explain why we cannot improve its performance by just swapping out the RSA signatures in [48] and replacing them with ECDSA. Finally, we construct a ECC-based VRF, and prove its security in Appendix B.

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).

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

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

and then compute the hash value β as

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

H is a cryptographic hash function (*e.g.,* SHA-256) while MGF is an IETF-standard cryptographic hash that produces outputs one bit shorter than the RSA modulus [22, Sec. 10.2] (*aka,* a "full domain hash" [25]). Notice that anyone can compute β given π.

Verifying. To verify that β is the VRF hash of α, first verify that $H(\pi) = \beta$ and then 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. Appendix C proves its security in the random oracle model.

4.1 VRF based on RSA

The original NSEC5 construction [48] was not described in terms of VRFs. However, it actually uses the VRF in Figure 1, which is based on RSA in the random oracle model. Notice that the VRF proof is simply a deterministic RSA signature (using [25]'s "full-domain hash" construction), and the VRF output is simply the cryptographic hash of the VRF proof. VRF verification amounts to an RSA verification of the VRF proof. We prove that this is a secure VRF in Appendix C.

Use with NSEC5. 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).

4.2 Why can't we just use ECDSA?

At this point, one would naturally wonder why we don't just replace the RSA signature in Figure 1 with an ECDSA signature. 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 while the "full-domain hash" RSA signature used in Figure 1 is *unique* given the public key PK, an ECDSA signature lacks this property. With randomized ECDSA signatures, the signature is computed using a random nonce, and so signatures are not unique given *only* the ECDSA public key PK. Moreover, even "deterministic" ECDSA [73] fails to provide uniqueness given *only* the ECDSA public key PK. With "deterministic" ECDSA, the signer derives the signing nonce from a keyed hash of the

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 onto $G - \{1\}$. 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 an ℓ-bit integer.

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

Hashing. 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 random nonce $k \in \{0, \ldots, q-1\}$.
(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. (Note that c may be shorter than a full-length exponent, because its length is determined by the choice of H_3). The VRF output $\beta = F_{SK}(\alpha)$ is computed by shortening γ^f with H_2. Thus

$$\pi = (\gamma, c, s) \qquad \beta = H_2(\gamma^f)$$

Notice that anyone can compute β given π.

Verifying. Given public key PK, verify that proof $\pi = (\gamma, c, s)$ corresponds to the input α and output β as follows:
(1) Compute $u = (PK)^c \cdot g^s$.
(Note: if everything is correct then $u = g^k$.)
(2) Given input α, hash it to obtain $h = H_1(\alpha)$.
Check that $\gamma \in E$.
Compute $v = (\gamma)^c \cdot h^s$.
(Note: if everything is correct then $v = h^k$.)
(3) Check that hashing all these values together gives us c from the proof. That is, check that:

$$c = H_3(g, h, PK, \gamma, u, v)$$

Finally, compute $\beta = H_2(\gamma^f)$.

Figure 2: A VRF that operates in a cyclic group G of prime order with generator g. We use a multiplicative group notation. This VRF adapts the Chaum-Pederson protocol [35] for proving that two cyclic group elements g^x and h^x have the same discrete logarithm x base g and h, respectively. Appendix B proves its security in the random oracle model, based on the *decisional Diffie-Hellman (DDH)* assumption, which roughly says that h^x looks random given the tuple (g, g^x, h).

message it is signing, but the symmetric key k to this hash is independent of the ECDSA public key PK. Thus, the signer could produce a different ECDSA signature just by choosing a different key k, and the verifier would never know the difference.

Why does this matter? If ECDSA signatures were used in the construction of Figure 1, then the VRF prover could produce any arbitrary number of valid VRF proofs π for a given input α and public key PK. This clearly violates the trusted uniqueness property of the VRF (Section 3.1). Per Section 3.3, trusted uniqueness is central to the strong integrity property of NSEC5. This is why we can't base NSEC5 on ECDSA signatures.

4.3 VRF based on Elliptic Curves.

We now see how to produce shorter NSEC5 responses using elliptic curves (ECC). Our starting point is construction of [41, 46]. We cannot, however, we use [41]'s construction as is. While [41] claimed their construction was also a VRF, they did not formally prove that it achieves the VRF properties from Section 3.1. In fact, we discovered that their construction (which has since been adopted by Google's Key Transparency project [11, 62]) has a critical flaw that allows a malicious prover to violate the VRF's trusted uniqueness property. This flaw has since been corrected as a result of our work [7, 8, 40].

Our VRF construction can be seen in Figure 2 and our formal proof of its security properties in Appendix B. It fixes the flaw of [41], without any downgrade in performance. On the contrary, since we provide a concrete (as opposed to asymptotic) security analysis as per the formulation of [26], we can optimize the VRF's parameters. Concretely, we can shorten the length of VRF proof π, by truncating value c in Figure 2 so that it is only ℓ bits long (and not $2 \cdot \ell$). This results in NSEC5PROOF records that are ℓ bits shorter.

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 (NIST P-256 [55, Sec. 3]), and Curve25519 [59] which has recently been proposed for use with DNSSEC [56, 77]. Each of these curves operates in finite field F_p where p is a 256-bit prime, and achieves a security level of $\ell = 128$ bits [27, 55].

Use with NSEC5. What response lengths do we get when we instantiate NSEC5 with the VRF in Figure 2 over 256-bit elliptic curves?

Each NSEC5 record will once again contain two hash outputs (each corresponding to β in Figure 2) along with a DNSSEC signature. We instantiate H_2 in Figure 2 with the function that outputs the x coordinate (abscissa) of a point (x, y) on the elliptic curve (where $x, y \in F_p$). Thus, each β will be 256-bits long.

We instantiate H_1 per Appendix A.

Next, observe that each NSEC5PROOF record will contain the proof value $\pi = (\gamma, c, s)$ from Figure 2. How long is π? If we instantiate the VRF using a 256-bit elliptic curve (*e.g.,* NIST P-256 or Ed25519), then s is 256 bits long. Meanwhile, γ is a point on the elliptic curve, which can be represented

example.com	A
bar.example.com	A
www.example.com	A
*.www.example.com	A

Figure 3: Example zone.

with $256 + 1$ bits using point compression.[4] Finally, we show (in Appendix B) c must be ℓ-bits long for an ℓ-bit security level. We therefore instantiate H_3 as the first 128 bits output by the SHA-256 hash function.

It follows that proof π will be $p = 256+1+\ell+256 = 513+\ell$ bits for a ℓ-bit security level; thus, $p = 641$ for a 128-bit security level. Achieving the same security level with RSA requires 3072-bit RSA, which results in NSEC5PROOFS that are about 5 times longer!

5 DESIGNING THE DNS PROTOCOL

To properly understand the performance of NSEC5, we must move beyond the clean and idealized model we 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. This is primarily due to the complex nature of a seemingly-unrelated issue: DNS wildcards [60, Section 7.2.1],[45, 61]. (Indeed, the treatment of DNS wildcards is so complex that RFC4592 [61] clarifying their use was issued nineteen years after the original DNS RFC1035 [65].) Thus, we start by digging into how NSEC3 handles wildcards. We then design the protocol that NSEC5 uses to deal with wildcards, and describe how it (1) uses a "wildcard bit" to shorten response lengths and (2) exploits precomputation to improve performance.

5.1 Wildcard and closest encloser proofs.

A wildcard record maps a set of queries to a particular response. For example, if the domain has a wildcard record for `*.example.com`, then queries for `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").

To see why wildcards matter, we use a running example. Suppose a DNS query for `a.b.c.example.com` is made to the example zone in Figure 3. 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` [61]. Next, `*.example.com`—the wildcard child of the closest encloser—is not in the zone. Thus, there is no *wildcard expansion* of `a.b.c.example.com`. The correct 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 it done, the nameserver must additionally prove:

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

5.2 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 [60]:

(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 $h_1 < h(*.\texttt{example.com}) < h_2$.

Thus, wildcards significantly impact performance: a single query can solicit up to three NSEC3 responses! (Figure 4.)Sometimes, fewer than three NSEC3 records are needed. For instance, 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 below and in Table 3.

Response length. Every query can elicit a response containing (up to) three NSEC3 records, each of which includes as DNSSEC signature (of length σ bits) and two hash values (each of length 2ℓ bits). Thus, the bitlength of the response can be estimated as

$$|\text{nsec3}| = 3(4\ell + \sigma) = 12\ell + 3\sigma \quad (1)$$

Resolver computations. The resolver must verify up to three DNSSEC signatures (on each NSEC3 record).

Nameserver computations. When regular NSEC3 is used, all responses are precomputed. When NSEC3 White Lies is used, responses are generated on the fly, so up to three NSEC3 records are signed in response to every query.

[4]The idea behind point compression is to represent a point with coordinates (x, y) using only its abscissa x (which is 256 bits long) and a single bit that indicates which square root (positive or negative) should be used for the ordinate y. Without point compression, both coordinates must be transmitted, for a total length of 256+256 bits. (Thus, without point compression our proof π would be $2 * 256 + 128 + 256 = 896$ bits long.) There has been some controversy over whether or not point compression is covered by a patent, and whether its use in DNSSEC corresponds to patent infringement [81]. However, as Bernstein [29] argues: "a patent cannot cover compression mechanisms [appearing in the paper by Miller in 1986 [64] that was] published seven years before the patent was filed." Moreover, new IETF specifications for elliptic curve digital signatures using Ed25519 also use point compression [56].

	online crypto at nameserver	verifications at resolver	max response length
NSEC	none	2 RRSIGs	2σ
NSEC3	none	3 RRSIGs	$3\sigma + 12\ell$
NSEC3 White Lies	1 RRSIG	3 RRSIGs	$3\sigma + 12\ell$
NSEC5	1 NSEC5PROOF	2 RRSIGs 2 NSEC5PROOFs	$2\sigma + 8\ell + 2p$

Table 3: Performance characteristics of NXDOMAIN responses for NSEC*. 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 p is the bitlength of an NSEC5PROOF.

5.3 Adding the wildcard bit to NSEC5.

In [45], however, Gieben and Mekking observed that wildcards could be dealt with 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 $h_1 = h($`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 need only 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 in practice [44]. However, we can use it with NSEC5, because NSEC5 records have the same structure as NSEC3 records.

NSEC5 uses the wildcard bit, so that up to two NSEC5 records (and two NSEC5PROOFs) are needed to respond to any query. (See Figure 5.)This has significant impact on response lengths:

Response lengths. 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 p bits). We can therefore estimate the total bitlength of the response as

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

Resolver computations. Resolvers need to verify two NSEC5PROOF records and up to two DNSSEC signatures (on each NSEC5 record).

5.4 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.Online crypto is only needed for the second NSEC5PROOF. The second NSEC5PROOF and NSEC5 record *cover* the next closer `c.example.com`. The NSEC5PROOF proves that β is a correct VRF hash of `c.example.com`. Meanwhile, the NSEC5 record has a pair of VRF hash outputs h_1, h_2 that must fall lexicographically before and after β. Importantly, h_1 and h_2 must *not* equal β. Also, β is unknown at the time that the NSEC5 record is prepared. As such, the NSEC5PROOF for β cannot be precomputed.

Thus NSEC5 only needs one online cryptographic computation when the nameserver responds to a query.[5]

6 PRACTICAL CONSIDERATIONS

NODATA Responses. Thus far, our exposition has been a clean and idealized model where 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. For instance, the AAAA record is for IPv6 addresses. Suppose the example zone in Figure 3 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* [20, 60], 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 the AAAA bit cleared. Notice that NODATA responses always use just one NSEC* record!

Privacy. Wildcards and types have only minor implications on NSEC5 privacy.

Consider a queried name (*e.g.,* `a.b.c.example.com`) that does not exist in the zone. Then, the NXDOMAIN response reveals the closest encloser's name (`example.com`) and types that exist in the zone (*e.g.,* A, AAAA, MX, NS), and also reveals if its wildcard child (`*.example.com`) exists in the zone. Meanwhile, if a queried name (*e.g.,* `www.example.com`) does exist in the zone, then the NODATA response reveals its all types (*e.g.,* A) present in the zone.

[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. Therefore, 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.

This means that NSEC5 ensures that an attacker can learn which types of a non-wildcard name (`example.com`) exist in the zone only if it (1) queries for the exact name (`example.com`) OR (2) queries for any longer name that contains it as a prefix (*e.g.,* `a.b.c.example.com`). In other words, the attacker must still enumerate the zone by brute force, sending an online query for every name (or longer name that contains it as a prefix) suspected to be in the zone.

Opt-out, key rollover. Because NSEC5 is so similar in structure to NSEC3, it also supports other important optimizations and procedures developed for DNSSEC. For instance, NSEC5 supports opt-out in the same way as NSEC3 [60]. Moreover, the NSEC5KEY can be rolled over using the same procedure to roll a ZSK [58]: the new NSEC5KEY record is published, then old NSEC5 records are replaced by NSEC5 records computed using the new NSEC5KEY, and finally the old NSEC5KEY is removed from the zone.

7 IMPLEMENTATION

We designed and implemented the two NSEC5 varients (RSA and ECC), extending existing DNS software. For the authoritative nameserver, we extended Knot DNS 1.6.4, 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 implementation supports the full spectrum of negative responses, (*i.e.,* NXDOMAIN, NODATA, Wildcard, Wildcard NODATA, and unsigned delegation). The authoritative implements the optimization that precomputes the NSEC5PROOFs matching each NSEC5 record (Section 5.4). 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. The code is deliberately modular, so that the Ed25519 curve [56] (which is not supported by OpenSSL v1.0.2j) could be used a drop-in replacement. Overall, we added approximately 9,000 lines of C code. We plan to make the source publicly available.**A "live" example from our implementation.** Figures 4 and 5 present "live" NXDOMAIN responses from our implementation, for NSEC3 and NSEC5 respectively. (Cryptographic values (hashes, proofs, and signatures) have been shortened and some data fields have been dropped.) To generate these responses, we signed a small `example.com` zone with NSEC3 using ECDSA-P256 (DNSSEC algorithm 13) and ECC-based NSEC5. Per Section 5.2, NSEC3 returns three records and their corresponding signatures. On the other hand, the wildcard bit used with NSEC5 allows us to return only two NSEC5 records and two NSEC5PROOFS (Section 5.4).

```
$ kdig +dnssec +multiline ddadasds.example.com
;; ->>HEADER<<- opcode: QUERY; status: NXDOMAIN; id: 22793
;; Flags: qr aa rd; QUERY: 1; ANSWER: 0; AUTHORITY: 8; ADDITIONAL: 1

;; QUESTION SECTION:
;; ddadasds.example.com.  IN A

;; AUTHORITY SECTION:
example.com.     3600 IN SOA dns1.example.com.
example.com.     3600 IN RRSIG SOA 13 2 3600 20170128184611
             ( 5134 example.com. nqiEgM+kVBDeBI== )

;; Matching record for hash of example.com --closest encloser;
0sc7qshrek878fcmnag1.example.com. 3600 IN NSEC3 1 0 0 AABB
             ( CPDHD7GK40NGDKRU8CQ8 NS SOA MX RRSIG DNSKEY NSEC3PARAM )
0sc7qshrek878fcmnag1.example.com. 3600 IN RRSIG NSEC3 13 3 3600
             ( 5134 example.com. 2JicIoTH3WkgAjbP/ehmTv== )

;; Covering record for hash of ddadasds.example.com --next closer record;
jftj44t4kqppke20mukr.example.com. 3600 IN NSEC3 1 0 0 AABB
             ( MSC7QSHREK878FCM8GD7 A AAAA RRSIG )
jftj44t4kqppke20mukr.example.com. 3600 IN RRSIG NSEC3 13 3 3600
             ( 5134 example.com. VfFQfho5sQ8QVWOqsrXyN6== )

;; Covering record for hash of *.ddadasds.example.com --wildcard record;
cpdhd7gk40ngdkru8cq8n.example.com. 3600 IN NSEC3 1 0 0 AABB
             ( J1VSBFDBU38SMLNJPIMM A AAAA RRSIG )
cpdhd7gk40ngdkru8cq8n.example.com. 3600 IN RRSIG NSEC3 13 3 3600
             ( 5134 example.com. lcDsoeVGuq3rvezN2oW74x== )

;; Received 773 B
```

Figure 4: NXDOMAIN response with NSEC3.

```
$ kdig +dnssec ddadasds.example.com
;; ->>HEADER<<- opcode: QUERY; status: NXDOMAIN; id: 18282
;; Flags: qr aa rd; QUERY: 1; ANSWER: 0; AUTHORITY: 8; ADDITIONAL: 1

;; QUESTION SECTION:
;; ddadasds.example.com.    IN A

;; AUTHORITY SECTION:
example.com.    3600 IN SOA dns1.example.com.
example.com.    3600 IN RRSIG SOA 16 2 3600
             ( 5137 example.com. kVfd4pgDmWMg== )

;; Matching record for hash of example.com --closest encloser;
;; Wildcard flag is not set;
ec2i1k1adn16bb9sbh1k.example.com. 86400 IN NSEC5 48566 0
             ( H4ETTRT2RNLVQA2DU6HM NS SOA MX RRSIG DNSKEY NSEC5KEY )
ec2i1k1adn16bb9sbh1k.example.com. 86400 IN RRSIG NSEC5 16 3 86400
             ( 5137 example.com. RbkKnf4MT/Fg== )

;; Covering record for hash of ddadasds.example.com --next closer record;
4vulla22dr6bo63j203c.example.com. 86400 IN NSEC5 48566 0
             ( C341KKJADV09N1BH2DJ0 A AAAA RRSIG )
4vulla22dr6bo63j203c.example.com. 86400 IN RRSIG NSEC5 16 3 86400
             ( 5137 example.com. KMrN9N+J9Rug== )

;; NSEC5PROOF records;
example.com.  3600 IN NSEC5PROOF 48566 ( AiZnaTPduKWyig )
ddadasds.example.com.  3600 IN NSEC5PROOF 48566 ( AzH6uKGjS+2FJf )

;; Received 834 B
```

Figure 5: NXDOMAIN response with NSEC5.

8 PERFORMANCE EVALUATION

We now evaluate the performance of NSEC5 and compare it against (plain) NSEC3 and online signing with NSEC3 White Lies (Section 2). We consider response length, query processing time at the recursive resolver and authoritative nameserver, and throughput, memory and CPU usage at the authoritative.

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.

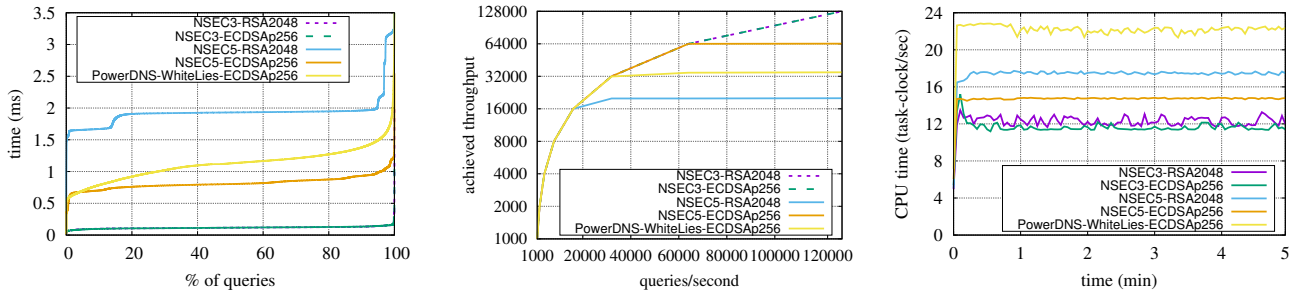

Figure 6: (Left) Query processing time at the authoritative nameserver per NXDOMAIN response. (Center) Throughput at the authoritative nameserver under stable query rate when all queries result in NXDOMAIN responses.(Right) CPU utilization (task-clock/second) for the different authoritative configurations at 32 Kqps query load and 24 threads.

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.

Zone. We test against a real Alexa-100 second-level-domain (SLD) zone that consists of about 1000 names.

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 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 at 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 (*e.g.,* A, AAAA, MX, NS records) as well as negative ones (NXDOMAIN) [5].

8.1 Response lengths.

We want DNSSEC responses to be short enough to fit into a single IP packet and to limit DDoS amplification (Section 1). Our measurements show 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 7. Figure 7 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 packet (Ethernet MTU). Meanwhile, ECC-based NSEC5 is much shorter (827 bytes, on average), easily fitting into a single IP packet, and is comparable to ECC-based NSEC3 (783 bytes).

Comparison to "legacy" NSEC3. Modern cryptographic recommendations mandate a security level of at least 112 bits [23]. Despite these recommendations, NSEC3 only supports (outdated) SHA1 as its hash function [60], for an (outdated) security level of $\ell = 80$ bits. (NSEC5 records use a $2\ell = 256$-bit hash outputs, for a $\ell = 128$-bit security level.) Also, most domains deploying DNSSEC still use 1024-bit RSA ($\sigma = 1024$ bits) [18, 78], for an (outdated) 80-bit security level [23]. 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)!

8.2 Nameserver performance.

Both NSEC5, and online signing with NSEC3 White Lies, prevent offline zone enumeration by requiring online public-key crypto computations at the nameserver. (See Table 3.) We now 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, 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. Figure 6-(left) presents

[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 focus our NSEC5 implementation effort on the more performant Knot DNS nameserver.

the results. Ignoring the tail of the plot (which can be attributed to delays in inter-process communication and other tasks running in the background), we see that the majority of queries are processed consistently close to an average time for each configuration. Plain NSEC3 (with RSA-2048 and ECDSA-P256) uses precomputed responses; as such, the nameserver can respond to queries in just 117μs and 116μs on average. 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. This is *faster* than the 1.12ms query processing time for the PowerDNS implementation of NSEC3 White Lies![7]

Throughput with purely negative traffic. Next, we consider aggregate query throughput. We used Dnsperf 2.1.1 [15], a popular open-source DNS performance evaluation tool, to issue negative queries at fixed rates from 1K to 128K queries per second (qps). Figure 6-(center) presents throughput results on a logarithmic scale.

Plain NSEC3 does not use online cryptographic computations, and so throughput scales easily to 128 Kqps and beyond. 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 [13]. ([13] finds a 2-3x gap in throughput between the Knot DNS and PowerDNS when serving DNSSEC-enabled zones.)Our NSEC3-ECC throughput results should be well above the needs of most zone operators. To put this in context, the A operator [6] reports an average negative query load per server that is roughly one order of magnitude smaller.

[7]Per footnote 5, PowerDNS caches and reuses NSEC3 records generated on-the-fly for the closest encloser and wildcard. By contrast, our NSEC5 implementation *precomputes* the closest-encloser records, rather than caching and reusing them. Thus, to fairly compare across implementations, we crafted the query load so that all queries could use the same records (served from cache) for all but the next-closer records (Section 5.1). Therefore, both NSEC5-ECC and NSEC3 White Lies perform a single online crypto computation at query time.

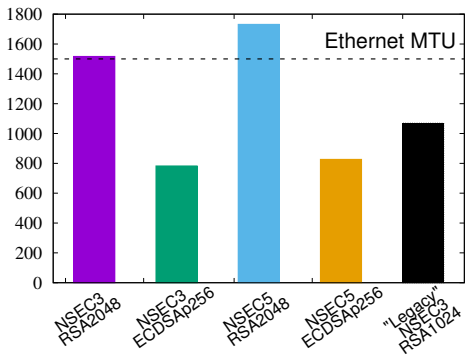

Figure 7: Average length for a single NXDOMAIN response (standard deviation $< 1\%$).

	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

Table 4: Memory footprint (MB) at the authoritative after loading the zone.

Throughput with mixed traffic. In practice, throughput should be even higher, because normal traffic should elicit positive responses (*e.g.,* signed A records), which are precomputed, in addition to NXDOMAIN responses. To demonstrate this, we tested ECC-based NSEC5 at a steady query rate of 32 Kqps using 4 (rather than 24) threads. When fewer than 50% of responses are NXDOMAIN, throughput remains steady at 32 Kqps. Meanwhile, purely NXDOMAIN traffic saturates throughput at 13 Kqps.

CPU utilization. CPU utilization is shown in Figure 6-(right). We used the Linux `perf_events` profiler to measure the `task-clock` time per second (shown on the y-axis of Figure 6-(right)), which reports the CPU time spent by a process across all threads. Since we use 24 threads, full utilization would correspond to a task-clock/second of 24. All measurements were taken over a 5 minute period (time shown on the x-axis) with 32 Kqps query load of purely NXDOMAIN traffic. From Figure 6-(center), we already know that a 32 Kqps query load causes throughput to deteriorate for RSA-based NSEC5 and PowerDNS's NSEC3 White Lies, but not for plain NSEC3 and ECC-based NSEC5. Considering the corresponding CPU utilization in Figure 6-(right), we see that plain NSEC3 has the lowest CPU utilization (roughly 50%, or `task-clock` time/second of about 12) while NSEC3-ECC is not too much higher. Meanwhile, NSEC3 White Lies (with PowerDNS) has the heaviest CPU utilization (roughly 95%, or `task-clock` time/second of about 23), mostly due to implementation differences between Knot DNS and PowerDNS. As a final note, we expect utilization to be lower in a setting tuned for maximum performance, since these results include the heavy logging necessary for our experiments.

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 5.4), and (ii) NSEC5 records use 256-bit hash values, while NSEC3 uses (outdated, less secure) 160-bit SHA1 hash values. Finally, the memory overhead for NSEC3 White Lies is tiny, because NSEC3 records are computed on the fly at query time.

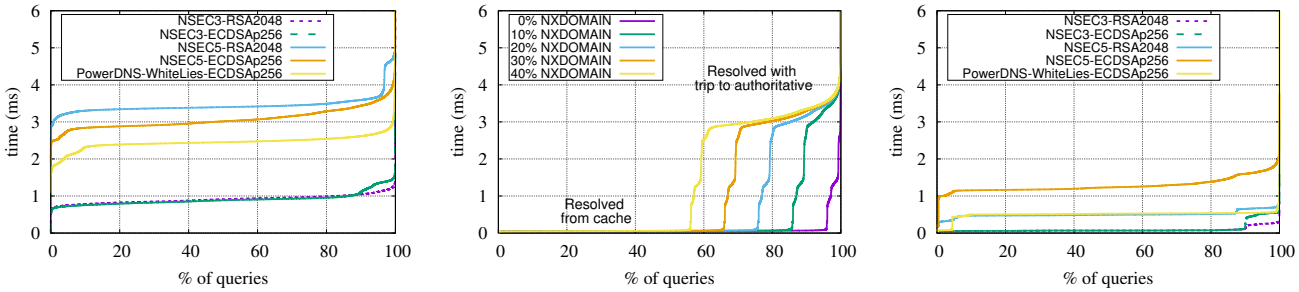

Figure 8: Overall query processing time at the recursive resolver and authoritative nameserver (left) per NXDOMAIN response across all configurations, and (center) for ECC-based NSEC5 under mixed (positive & NXDOMAIN) traffic. (right) Validation time per NXDOMAIN response at the recursive resolver for all configurations.

8.3 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 8-(left) reports the overall query processing time per NXDOMAIN response, as observed by a stub resolver. This measurement 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 authoritative nameserver.

Figure 8-(left) 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. (Compare this to processing time at the authoritative nameserver alone, which is orders of magnitude faster for plain NSEC3). Naturally, overall processing time for plain NSEC3 is fastest (about 1ms); again, this follows because plain NSEC3 does not require online crypto at the authoritative 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).

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 8-(center) 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 [57]. Naturally, NSEC5 only affects performance for negative queries; everything else is validated from cache in minimal time.

Validation time. Finally, we zoom in on performance at the recursive resolver by considering only the time required for validating responses. (This excludes processing at the nameserver, latency to the nameserver, packet processing at the recursive, *etc.*).

Figure 6-(right) shows that cryptographic validation NSEC5-RSA is faster than NSEC5-ECC. (This is natural: RSA verification is well known to be faster than ECDSA verification.)

Next, consider the two plain NSEC3 configurations. Figure 6-(center) shows that most queries are validated in microseconds; meanwhile, the top 11% of queries (on the right side of the figure) take seconds to validate. The reasoning for this subtle. Because we issue 100,000 queries for a zone that only has 1000 names, our recursive resolver eventually collects all the NSEC3 records for the zone. (In other words, it enumerates the zone.) Once this happens, the authoritative nameserver begins sending NSEC3 records that the recursive resolver has already cached. Instead of cryptographically validating these NSEC3 records from scratch, the resolver simply takes a few microseconds to retrieve the cached NSEC3 record. Thus, the excellent validation performance of plain NSEC3 follows because we make a large number of queries to the same small zone. In a live system that queries multiple zones, this behavior is likely to be less significant.

Now consider the validation performance for NSEC3 White Lies. With White Lies, a fresh NSEC3 record is generated for every query, so the recursive will never be able to collect all the NSEC3 records for the zone. (That is, will never be able to enumerate the zone unless it queries specifically for all names in it!) Thus, this excellent validation performance we observed for plain NSEC3 is not possible with NSEC3 White Lies. Analogous reasoning shows it is also not possible with any other approach that prevents zone enumeration, including NSEC5.

Thus, it is most sensible to compare NSEC5's validation performance to that of NSEC3 White Lies. Figure 8-(right) shows that validation for NSEC3 White-Lies (0.5ms) is faster

than for NSEC5-ECC implementation (1.2ms). Digging into this result, we found that it is due to (1) parsing and logging the different parts of the NSEC5 response (*e.g.,* the NSEC5PROOF), (2) fetching the NSEC5KEY, and (3) a performance gap between our (unoptimized) ECC-based VRF verification and the highly-optimized OpenSSL verification of ECDSA.

Remark: Speedups with Ed25519? Finally, we note that our NSEC5 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 [77] elliptic curve. We leave this to future work.

9 NSEC5 VS. RECENT INNOVATIONS

We consider the relationship between NSEC5 and some recent DNS innovations.

Aggressive negative caching (draft-ietf-dnsop-nsec-aggressiveuse) [42]: A new proposal, that is in the process of being standardized, calls for *aggressive caching* of NSEC* records at resolvers. The idea is to reuse cached NSEC* records to answer queries that are *different* from the original query that elicited the NSEC* record. (The original DNSSEC specifications [21] do not allow this.) To see how this works, suppose the zone in Figure 3 used (plain) NSEC and suppose we sent a type A query for `foo.example.com`. The response would contain an NSEC record that (1) attests that no names exist between `bar.example.com` and `www.example.com`, and (2) has a type bitmap with the type A bit set and type AAAA, NS, MX, *etc.* bits cleared. Then, aggressive negative caching allows resolvers to use the cached NSEC record to infer that:

(1) Other names covered by the NSEC record do not exist in the zone (NXDOMAIN for *e.g.,* `qqq.example.com`).
(2) Other types matching the NSEC record do not exist in the zone (NODATA for `bar.example.com` for types *e.g.,* AAAA, NS, MX).

This first item treats offline zone enumeration as feature, rather than a bug. In other words, it exploits the fact that resolvers can make offline inferences about the names covered by an NSEC/NSEC3 record. It optimizes DNSSEC performance by cutting down on the number of queries sent from resolver to nameserver. (For instance, the fast response validation behavior we observed for plain NSEC3 in Figure 8-(right) would also translate to a reduce number of queries.) However, this performance optimization is obviated by *any* scheme that prevents offline zone enumeration, including NSEC3 White Lies and NSEC5, because these schemes *necessarily* prevent resolvers from making offline inferences about the names present or absent in the zone. Meanwhile, the second item optimizes performance (reducing queries from resolver to nameserver) for all the schemes including NSEC5.

RFC8020 [32]. RFC8020 is a new standard that states that NXDOMAIN for a query (`c.example.com`) implies that names deeper in the DNS hierarchy (*e.g.,* `b.c.example.com`) also do not exist. This allows resolvers to cache the NXDOMAIN response for `c.example.com` and reuse it to answer a later query for *e.g.,* `b.c.example.com`. All the NSEC* variants we have considered thus far, including NSEC5, can benefit from this performance optimization.

Black Lies (draft-valsorda-dnsop-black-lies [79]). There is a (concurrent) NSEC* proposal that leverages the fact that NODATA responses are short. Black Lies is an online-signing solution that answers each negative query with an NODATA response, even if the "correct" response is NXDOMAIN. (Hence, the Black Lie.) For example, suppose the zone in Figure 3 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.

Black Lies comes with some caveats. Most importantly, it is an online-signing solution (per Tables 1,2) that requires the nameserver to know the secret zone-signing key (ZSK). Thus, it fails to provide strong integrity. Moreover, because Black Lies gives a NODATA response when the "correct" response is NXDOMAIN, it obviates the performance optimization of RFC8020 [32]. Also, Black Lies thwarts any diagnostic or security tool (*e.g.,* [38, 74]) that uses NXDOMAIN responses to infer that a name definitely does not exist in the zone.

10 SUMMARY: WHY USE NSEC5?

The key advantage of NSEC5 is that it (1) stops offline zone enumeration while (2) providing *strong integrity* even if the zone's authoritative nameserver is compromised. By contrast, DNSSEC's online signing solutions (NSEC3 White Lies [44], Minimally-Covering NSEC [86], Black Lies [79]) stops offline zone enumeration by trusting the nameserver with the secret zone-signing key (ZSK); thus compromising the nameserver compromises the integrity of the zone (Table 2).

[48] proved that providing integrity and preventing offline zone enumeration *necessarily* require the nameserver to perform one online public-key crypto computation 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 8.2 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 [6] 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 mixed traffic, with both positive and negative queries) and a single server (rather than a cluster of nameservers, a more common deployment configuration).

With ECC-based NSEC5, the overall processing time for an negative query (from stub resolver, to recursive resolver, to authoritative nameserver) is only 30% longer that of online signing with NSEC3 White Lies (using the PowerDNS implementation). It may be possible to reduce this performance gap with an optimized implementation, since the nature and number of cryptographic operations in the two configurations is similar. Moreover, our implementation is for the NIST P-256 elliptic curve; further speedups might be possible by moving to the Ed25519 curve [77]. (Doing this requires no modifications to ECC-based VRF of Figure 2. Response length results for an Ed25519 instantiation would be identical to those in Section 8.1, because both P-256 and Ed25519 are 256-bit curves.)

Thus, we believe that NSEC5 can be a practical solution for zone operators that care about protecting sensitive information (names of hosts, servers, routers, IoT devices, DANE certificates [54], *etc.*) from offline zone enumeration attacks. Meanwhile, operators that don't care about zone enumeration should just use plain NSEC3. Moreover, for zones that currently use online signing with NSEC3 White Lies, moving to NSEC5 seems like a win-win scenario: roughly the same (if not better) performance, and no need to store the sensitive secret ZSK at the authoritative nameserver.

11 THE TRANSITION TO NSEC5

We conclude with a discussion of the elephant in the room. How can today's DNSSEC transition to NSEC5?

The DNS community has faced this problem before. First, the NSEC3 specification [60] came out after the earliest deployments of DNSSEC [68], and so resolvers and nameservers had to transition from NSEC to NSEC3 [60, Section 10.4]. Second, there is currently a proposal to transition from RSA to ECDSA signatures over the NIST P-256 elliptic curve [81]. Third, a desire to avoid NIST-specified curves [30] and to have short DNSSEC responses, is motivating the community to consider transitioning to digital signatures over Edwards elliptic curves [77, 88]. Fourth, there is also the DPRIVE initiative that seeks to add confidentiality to DNS transactions, to mitigate concerns surrounding pervasive network monitoring [3]. Given that other transitions may be on the horizon, this might also be a good time to consider transitioning to NSEC5.

11.1 The mechanics of the transition.

We believe that the 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 [9]). 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 [21, 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.

New algorithm numbers could also be used to transition to NSEC5. There are two ways [58, Sec 4.1.4] to transition from an old algorithm number to a new one.

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 [21, 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.[8] If online signing (*e.g.,* NSEC3 White Lies) is supported in parallel with NSEC5, then the nameserver must hold the secret ZSK key, and thus NSEC5 loses its strong integrity guarantees. On the other hand, the liberal approach is unlikely to be used in a transition until a majority of resolvers support NSEC5. However, given that resolvers might soon be upgraded to add support for Edwards curves, now might also be a good time to consider adding support for NSEC5.

ACKNOWLEDGEMENTS

We thank innumerable DNS practitioners for pushing us to develop a more performant version of NSEC5. We also thank Asaf Ziv, Sachin Vasant, Ondrej Sury and Tomofumi Okubo for earlier collaborations on NSEC5. This research was supported, in part, by NSF grants 1012798, 1012910 and 5245250 and a gift from Verisign Labs.

REFERENCES

[1] 2011. Microsoft Security Bulletin MS11-058 - Critical: Vulnerabilities in DNS Server Could Allow Remote Code Execution (2562485). https://technet.microsoft.com/library/security/ms11-058. (August 9 2011).
[2] 2014. The Heartbleed Bug. http://heartbleed.com/. (2014).
[3] 2015. DNS PRIVate Exchange Working Group Charter (DPRIVE). (2015). https://datatracker.ietf.org/doc/charter-ietf-dprive/.

[8] This also suggests that algorithm negotiation [51] may be less helpful in a transition to NSEC5—a zone-enumeration attacker can simply negotiate to speak NSEC3.

[4] 2015. Microsoft Security Bulletin MS15-127 - Critical: Security Update for Microsoft Windows DNS to Address Remote Code Execution (3100465). https://technet.microsoft.com/en-us/library/security/ms15-127.aspx. (December 8 2015).

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

[6] 2017. A Root server raw data. http://a.root-servers.org/raw-data/index.html. (2017).

[7] 2017. A CONIKS implementation in Golang: Issue 175: Uniqueness of VRF is violated. https://github.com/coniks-sys/coniks-go/issues/175. (April 2017).

[8] 2017. Google Key Transparency Project: Issue 567: Uniqueness of VRF is violated. https://github.com/google/keytransparency/issues/567. (April 2017).

[9] 2017. IANA Domain Name System Security (DNSSEC) Algorithm Numbers http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml. (2017).

[10] 2017. Kali Tools: DNSRecon. http://tools.kali.org/information-gathering/dnsrecon. (2017).

[11] 2017. Key Transparency. (2017). https://github.com/google/keytransparency.

[12] 2017. Knot DNS. https://www.knot-dns.cz/. (2017).

[13] 2017. Knot DNS: Benchmark. https://www.knot-dns.cz/benchmark/. (2017).

[14] 2017. nmap: dns-nsec-enum. https://nmap.org/nsedoc/scripts/dns-nsec-enum.html. (2017).

[15] 2017. Nominum Measurement Tools. http://www.nominum.com/measurement-tools/. (2017).

[16] 2017. nsec3map: John the Ripper plugin. https://github.com/anonion0/nsec3map. (2017).

[17] 2017. PowerDNS. https://www.powerdns.com/. (2017).

[18] 2017. Verisign Labs SecSpider: Global DNSSEC deployment tracking. (2017). http://secspider.verisignlabs.com/.

[19] Brian Aitken. 2011. Interconnect Communication MC / 080:DNSSEC Deployment Study. http://stakeholders.ofcom.org.uk/binaries/internet/domain-name-security.pdf. (2011).

[20] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. 2005. *RFC 4034: Resource Records for the DNS Security Extensions*. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc4034.

[21] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. 2005. *RFC 4035: Protocol Modifications for the DNS Security Extensions*. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc4035.

[22] J. Staddon B. Kaliski. 1998. *RFC 2437: PKCS #1: RSA Cryptography Specifications, Version 2.0*. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc2437.

[23] Elaine Barker and Quynh Dang. 2015. Recommendation for Key Management - Part 3 Application-Specific (Revised). NIST Special Publication 800-57. (January 2015).

[24] Jason Bau and John C. Mitchell. 2010. A Security Evaluation of DNSSEC with NSEC3. In *NDSS*.

[25] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*. ACM, 62–73.

[26] Mihir Bellare and Phillip Rogaway. 1996. The Exact Security of Digital Signatures - HOw to Sign with RSA and Rabin. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*. 399–416. https://doi.org/10.1007/3-540-68339-9_34

[27] Daniel J Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography-PKC 2006*. Springer, 207–228.

[28] Daniel J. Bernstein. 2011. NSEC3 Walker. http://dnscurve.org/nsec3walker.html. (2011).

[29] D. J. Bernstein. 2017. Irrelevant patents on elliptic-curve cryptography. http://cr.yp.to/ecdh/patents.html (Accessed 1/15/2016). (2017).

[30] Daniel J Bernstein, Tanja Lange, and Ruben Niederhagen. 2016. Dual EC: A standardized back door. In *The New Codebreakers*. Springer, 256–281.

[31] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*. 514–532. https://doi.org/10.1007/3-540-45682-1_30

[32] S. Bortzmeyer and S. Huque. 2005. *RFC 8020: NXDOMAIN: There Really Is Nothing Underneath*. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc8020.

[33] Colin Boyd, Paul Montague, and Khanh Nguyen. 2001. Elliptic Curve Based Password Authenticated Key Exchange Protocols. In *Information Security and Privacy*, Vijay Varadharajan and Yi Mu (Eds.). Lecture Notes in Computer Science, Vol. 2119. Springer Berlin Heidelberg, 487–501. https://doi.org/10.1007/3-540-47719-5_38

[34] Melissa Chase and Anna Lysyanskaya. 2007. Simulatable VRFs with Applications to Multi-theorem NIZK. In *CRYPTO'07*. 303–322. https://doi.org/10.1007/978-3-540-74143-5_17

[35] David Chaum and Torben Pryds Pedersen. 1992. Wallet databases with observers. In *Advances in Cryptology-CRYPTO'92*. Springer, 89–105.

[36] Jean-Sébastien Coron. 2000. On the Exact Security of Full Domain Hash. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings (Lecture Notes in Computer Science)*, Mihir Bellare (Ed.), Vol. 1880. Springer, 229–235. https://doi.org/10.1007/3-540-44598-6_14

[37] J. Damas, M. Graff, and P. Vixie. 2013. *RFC 6891: Extension Mechanisms for DNS (EDNS(0))*. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6891.

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

[39] Dennis Fisher. 2012. Final Report on DigiNotar Hack Shows Total Compromise of CA Servers. Threatpost. https://threatpost.com/final-report-diginotar-hack-shows-total-compromise-ca-servers-103112/77170/. (2012).

[40] Matthew Franklin and Haibin Zhang. 2012 (updated 2017). *Unique ring signatures: A practical construction*. Technical Report 2012/577. ePrint Cryptology Archive.

[41] Matthew Franklin and Haibin Zhang. 2013. Unique ring signatures: A practical construction. In *Financial Cryptography and Data Security*. Springer, 162–170.

[42] K. Fujiwara, A. Kato, and W. Kumari. 2016. *draft-ietf-dnsop-nsec-aggressiveuse: Aggressive use of NSEC/NSEC3*. Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/draft-ietf-dnsop-nsec-aggressiveuse.

[43] Steve Gibson. 2002. *Distributed Reflection Denial of Service (DrDoS) Attacks*. Technical Report. Gibson Research Corporation.

[44] R. Gieben and W. Mekking. 2014. *RFC 7129: Authenticated Denial of Existence in the DNS*. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7129.

[45] R. Gieben and W. Mekking. 2015. draft-gieben-nsec4-00:DNS Security (DNSSEC) Authenticated Denial of Existence. (2015). https://tools.ietf.org/html/draft-gieben-nsec4-00.

[46] Eu-Jin Goh and Stanislaw Jarecki. 2003. A Signature Scheme as Secure as the Diffie-Hellman Problem. In *Advances in Cryptology - EUROCRYPT 2003*. 401–415. http://dx.doi.org/10.1007/3-540-39200-9_25

[47] Sharon Goldberg. 2014. NSEC5: Provably Preventing DNSSEC Zone Enumeration. In *DNS Operations Analysis and Research (DNS OARC) Fall 14 Workshop*. Los Angeles.

[48] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. 2015. NSEC5: provably preventing DNSSEC zone enumeration. In *NDSS'15*. https://eprint.iacr.org/2014/582.pdf.

[49] S. Goldberg, D. Papadopoulos, and J. Vcelak. 2017. draft-goldbe-vrf: Verifiable Random Functions. (2017). https://datatracker.ietf.org/doc/draft-goldbe-vrf.

[50] Amir Herzberg and Haya Shulman. 2013. Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all. org. In *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE, 224–232.

[51] Amir Herzberg and Haya Shulman. 2014. Negotiating DNSSEC Algorithms over Legacy Proxies. In *Proceedings of the 13th International Conference on Cryptology and Network Security - Volume 8813*. Springer-Verlag New York, Inc., New York, NY, USA, 111–126. https://doi.org/10.1007/978-3-319-12280-9_8

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

[53] Stacey Higginbotham. 2013. Anatomy of a hack: How the SEA took down the NYT and Twitter. https://gigaom.com/2013/08/27/anatomy-of-a-hack-how-the-sea-took-down-the-nyt-and-twitter/. (August 27 2013).

[54] P. Hoffman and J. Schlyter. 2012. *RFC 6698: The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSAC.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6698.

[55] P. Hoffman and W.C.A. Wijngaards. 2012. *RFC 6605: Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6605.

[56] S. Josefsson and N. Moeller. 2016. draft-irtf-cfrg-eddsa: Edwards-curve Digital Signature Algorithm (EdDSA). (2016). https://datatracker.ietf.org/doc/draft-irtf-cfrg-eddsa.

[57] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. 2001. DNS performance and the effectiveness of caching. In *Proceedings of the 1st ACM SIGCOMM Internet Measurement Workshop, IMW 2001*. 153–167. https://doi.org/10.1145/505202.505223

[58] O. Kolkman, W. Mekking, and R. Gieben. 2012. *RFC 6781: DNSSEC Operational Practices, Version 2.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6781.

[59] A. Langley, M. Hamburg, and S. Turner. 2016. *RFC 7748: Elliptic Curves for Security.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7748.

[60] B. Laurie, G. Sisson, R. Arends, and D. Blacka. 2008. *RFC 5155: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc5155.

[61] E. Lewis. 2006. *RFC 4592: The Role of Wildcards in the Domain Name System.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc4592.

[62] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. 383–398. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara

[63] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. 1999. Verifiable Random Functions. In *FOCS*. IEEE Computer Society, 120–130.

[64] Victor Miller. 1986. Use of elliptic curves in cryptography. In *Advances in Cryptology (CRYPTO'85 Proceedings)*. Springer, 417–426.

[65] P. Mockapetris. 1987. *RFC 1035: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc1035.

[66] Moni Naor and Asaf Ziv. 2015. Primary-secondary-resolver membership proof systems. In *Theory of Cryptography*. Springer, 199–228. https://eprint.iacr.org/2014/905.

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

[68] Eric Osterweil, Daniel Massey, and Lixia Zhang. 2007. Observations from the DNSSEC Deployment. In *The 3rd workshop on Secure Network Protocols (NPSec)*.

[69] Eric Osterweil, Daniel Massey, and Lixia Zhang. 2009. Availability Problems in the DNSSEC Deployment. (2009). http://irl.cs.ucla.edu/talks/2009-05-RIPE-PMTU.pptx.

[70] Eric Osterweil, Daniel Massey, and Lixia Zhang. 2009. Deploying and Monitoring DNS Security (DNSSEC). In *Twenty-Fifth Annual Computer Security Applications Conference, ACSAC 2009, Honolulu, Hawaii, 7-11 December 2009*. IEEE Computer Society, 429–438. https://doi.org/10.1109/ACSAC.2009.47

[71] Eric Osterweil, Danny McPherson, and Lixia Zhang. 2014. The Shape and Size of Threats: Defining a Networked System's Attack Surface. In *22nd IEEE International Conference on Network Protocols, ICNP 2014, Raleigh, NC, USA, October 21-24, 2014*. IEEE Computer Society, 636–641. https://doi.org/10.1109/ICNP.2014.101

[72] Eric Osterweil, Michael Ryan, Daniel Massey, and Lixia Zhang. 2008. Quantifying the operational status of the DNSSEC deployment. In *Proceedings of the 8th ACM SIGCOMM Internet Measurement Conference, IMC 2008, Vouliagmeni, Greece, October 20-22, 2008*, Konstantina Papagiannaki and Zhi-Li Zhang (Eds.). ACM, 231–242. https://doi.org/10.1145/1452520.1452548

[73] T. Pornin. 2013. *RFC 6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6979.

[74] Root Server System Advisory Committee (RSSAC). 2014. *RSSAC002: RSSAC Advisory on Measurements of the Root Server System.* Technical Report. ICANN. https://www.icann.org/en/system/files/files/rssac-002-measurements-root-20nov14-en.pdf.

[75] Marcos Sanz. 2004. DNSSEC and the Zone Enumeration. European Internet Forum: http://www.denic.de/fileadmin/public/events/DNSSEC_testbed/zone-enumeration.pdf. (October 2004).

[76] M. Sivaraman, S. Kerr, and L. Song. 2015. *draft-muks-dns-message-fragments-00: DNS message fragments.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/draft-muks-dns-message-fragments-00.

[77] O. Sury and R. Edmonds. 2016. *draft-ietf-curdle-dnskey-ed25519: Ed25519 for DNSSEC.* Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/draft-ietf-curdle-dnskey-ed25519/.

[78] Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, and Nadia Heninger. 2015. Factoring as a Service. Cryptology ePrint Archive, Report 2015/1000. (2015). http://eprint.iacr.org/2015/1000.

[79] F. Valsorda and O. Gudmundsson. 2016. *draft-valsorda-dnsop-black-lies: Compact DNSSEC Denial of Existence or Black Lies (expired).* Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/draft-valsorda-dnsop-black-lies/.

[80] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. 2014. DNSSEC and its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *IMC'14*. ACM, 449–460.

[81] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. 2015. Making the Case for Elliptic Curves in DNSSEC. *ACM SIGCOMM Computer Communication Review* 45, 5 (2015), 13–19.

[82] Jan Vcelak. 2015. NSEC5, DNSSEC Authenticated Denial of Existence. In *Security Area Advisory Group (SAAG) at IETF'92*. Dallas.

[83] J. Vcelak, D. Papadopoulos, and S. Goldberg. 2015. draft-vcelak-nsec5:NSEC5, DNSSEC Authenticated Denial of Existence. (2015). https://datatracker.ietf.org/doc/draft-vcelak-nsec5.

[84] Matthaus Wander. 2016. nsec3breaker. https://www.vs.uni-due.de/trac/dnssec. (2016).

[85] Matthaus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis. 2014. GPU-Based NSEC3 Hash Breaking. In *IEEE Symp. Network Computing and Applications (NCA)*.

[86] S. Weiler and J. Ihren. 2006. *RFC 4470: Minimally Covering NSEC Records and DNSSEC On-line Signing.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc4592.

[87] Hao Yang, Eric Osterweil, Daniel Massey, Songwu Lu, and Lixia Zhang. 2011. Deploying Cryptography in Internet-Scale Systems: A Case Study on DNSSEC. *IEEE Trans. Dependable Sec. Comput.* 8, 5 (2011), 656–669. https://doi.org/10.1109/TDSC.2010.10

[88] D. York, O. Sury, P. Wouters, and O. Gudmundsson. 2016. *draft-york-dnsop-deploying-dnssec-crypto-algs: Observations on Deploying New DNSSEC Cryptographic Algorithms.* Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/draft-york-dnsop-deploying-dnssec-crypto-algs/.

A 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 can work for both curves we have considered here: NIST P-256 and Curve25519.

One very lightweight technique was proposed in [31] and, at a high level, 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 $h = H(\alpha||i)$, where H is a standard cryptographic hash function, *e.g.,* SHA-256, and $||$ is concatenation. Then,

if $h^3 + ax + b$ is a quadratic residue (that is, h is the valid x-coordinate of a point on the curve) output the point $(h, (h^3 + ax + b)^{1/2})$ raised to the power of cofactor f. 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 (because only one y is chosen for a random x), but that does not materially change the proofs of security (specifically, in Claims B.4 and B.5, the running time for simulating queries to H_1 doubles).

As first shown in [33], 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, we stress that this attack is not relevant in the context of NSEC5. The only value that is hashed in the query phase is the queried name α itself, which is already known to the adversary.

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.

B.1 Proof sketches.

We start with a sketch of the proofs of three properties: uniqueness, psuedorandomness, and collision resistance. We define and prove them formally after this brief informal sketch. For this purposes of this sketch, 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

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

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

$$c_1 \equiv \frac{\log_g u - \log_h v}{x - \log_h \gamma_1} \pmod{q}. \quad (3)$$

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 (4) for a given $(g, h, g^x, \gamma, u, v)$, *regardless* of s. 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 (3) is negligible.[9] Thus, we have arrived at our contradiction.

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

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. 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.

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 $\alpha_1 \neq \alpha_2$. Assume H_2 is a τ-to-1 function. Since raising to the power x is a permutation, for every h_1, there are at most τ possible h_2 values that can cause a collision. Since h_1 and h_2 are obtained via random oracle queries, a pair that causes a collision is unlikely to be found after Q_H queries to H_1, as long as G is larger than $\tau Q_H^2/2$.

B.2 Full Proofs

We now expand on the sketches above to prove that the construction in Section 4.3 is a secure VRF. It suffices to prove three properties: Trusted Uniqueness (see [66, Definition 10]), Selective Pseudorandomness (see [66, Definition 11]), and Collision-Resistance (not formally discussed in [66], but mentioned in the proof of Theorem 4). Sufficiency of these three properties for constructing NSEC5 follows from [66, Theorem 4]. We discuss each property in turn.

We model the hash functions H_1 and H_3 as random oracles. We use notation $\mathsf{Ver}_{PK}(\alpha, \beta, \pi)$ 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.

B.2.1 Uniqueness. Recall that uniqueness requires that there should be only one provable VRF output β for every input α; *trusted* uniqueness limits this requirement to only the case when the public key is valid.

Following tradition of the VRF literature, Naor and Ziv [66, Definition 10]) define uniqueness unconditionally: that is, for a validly generated public key, each input α to the VRF has at most one hash output β that can be proven to be correct. However, the construction in Section 4.3 satisfies it only computationally: more than one hash output y may exist, but only one valid β—the one produced by $F_{SK}(\alpha)$—can be proven correct by any computationally bounded adversary, even given the secret key. We are not aware of any prior work defining this relaxation of the uniqueness property, although Chase and Lysyanskaya [34] mention that such a relaxation can be defined. We therefore define it here. Our definition is in terms of concrete, rather than asymptotic security, because concrete security enables us to set length parameters.

Definition B.1. (Computational Trusted Uniqueness.) A VRF satisfies (Q_H, ϵ)-trusted uniqueness if for all adversaries A that make at most Q_H queries to the random oracle, for a validly chosen key pair (PK, SK), the probability that the adversary can come up with an incorrect output $\beta_1 \neq F_{SK}(\alpha)$

and a proof for this β_1 is less than ϵ: namely,

$$\Pr[A(PK, SK) \to (\alpha, \beta_1, \pi_1) \text{ s.t. } \beta_1 \neq F_{SK}(\alpha) \text{ and } \mathsf{Ver}_{PK}(\alpha, \beta_1, \pi_1) = 1] \leq \epsilon\,.$$

We now prove that the VRF satisfies Definition B.1 based on the randomness of the oracle H_3. (Note: this proof does not rest on any computational assumptions or on programming a random oracle.)

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

Note that the quantitative bound on ϵ 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. 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 $\mathsf{Ver}(\alpha, \beta_1, \pi_1) = 1$ with probability ϵ. We will show that $\epsilon \leq (Q_H + 1)/\min(q/2, \rho)$, where q is the order of the group G and $\rho = |\text{range}(H_3)|$.

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 $\mathsf{Ver}(\alpha, \beta_1, \pi_1) = 1$. The verification function Ver ensures that $\gamma_1 \in E$ and computes $h = H_1(\alpha)$ and

$$u = g^s PK^c$$
$$v = h^s \gamma_1^c.$$

Because the VRF parameters and public keys are trusted, it follows that 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 two equations are guaranteed to be 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 $G = \{b \in E \,|\, b^q = 1\}$. 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:

$$\hat{u} = \hat{g}^s \hat{PK}^c$$
$$\hat{v} = \hat{h}^s \hat{\gamma}_1^c.$$

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 the logarithm of the second equation base $\hat{h}$, . Solving these for s we get

$$\log_{\hat{g}} \hat{u} - cx \equiv s \pmod{q}$$

$$\log_{\hat{h}} \hat{v} - c \log_{\hat{h}} \hat{\gamma}_1 \equiv s \pmod{q}$$

which implies that

$$c \equiv \frac{\log_{\hat{g}} \hat{u} - \log_{\hat{h}} \hat{v}}{x - \log_{\hat{h}} \hat{\gamma}_1} \pmod{q} \tag{4}$$

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 (4) given g, g^x, h, γ_1, u, and v.

Recall that for verification to pass,

$$c = H_3(g, h, g^x, \gamma_1, u, v)\,.$$

Note that the contents of the query to H_3 contains every value in the right hand side of equation (4), 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 $|\text{range}(H_3)|$. 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 range(H_3). Else (i.e., if $q < \rho$), think of reducing every element in 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 [66, Definition 10]. Thus, the proof that NSEC5, when constructed from the VRF of Figure 2, satisfies the soundness property in [66, Theorem 4] needs a slight change, as follows. The proof in [66] 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 [66]) and proof π for the VRF as part of its soundness-violating output on an input α (called x in [66]). 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 to $\epsilon - (Q_H + 1)/\min(q/2, \rho)$.

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 is 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 $PK \in E$.
- 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 also 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 $PK \in E$. The identifier of E must also be unambiguous in the sense that the adversary should not be allowed to choose the mapping from the group E to its identifier after seeing c.

B.2.2 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.

We require a slight modification to the notions of pseudorandomness and selective pseudorandomness from [66, Definition 11]: instead of being indistinguishable from a random bit string, the output of our VRF is indistinguishable from a truncation of a random element of $G - \{1\}$, i.e., from the distribution $H_2(U_G)$, where U_G is the uniform distribution on $G - \{1\}$. Our definitions are thus as follows.

Definition B.3. (Pseudorandomness) A VRF satisfies (t, Q_H, Q_P, ϵ) 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, whose total number of random oracle queries is bounded by Q_H and total number of Π and F queries is bounded by Q_P, can distinguish the following two games with advantage more than ϵ. In the both games, VRF keys (PK, SK) are honestly generated, and $D(PK)$ gets to query Π_{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 Π 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 Π_{SK} and F_{SK} (except on α^*), D outputs one bit indicating which game D thinks it is playing.

The slightly weaker notion of *selective* pseudorandomness is defined the same way, except D has to choose α^* before any queries and before seeing PK.

Pseudorandomness of our VRF depends on the following assumption about the group G and generator g, known as the *(t, ϵ)-DDH (Decisional Diffie-Hellman) Assumption*: for any adversary C whose description size and running time are bounded by t, the difference in probabilities (where the probabilities are over a random choice of $h, h' \in G - \{1\}$ and $x \in \{1, \ldots, q\}$) that $C(g^x, h, h^x) = 1$ and $C(g^x, h, h') = 1$ is at most ϵ. (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 now prove that our VRF satisfies both pseudorandomness and selective pseudorandomness. 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, ϵ)-DDH assumption, for any Q_H, Q_P, the VRF satisfies $(t', Q_H, Q_P, \epsilon')$ selective pseudorandomness for output distribution $H_2(U_G)$, for $t' \approx t$ (minus the time for $\Theta(Q_H + Q_P)$ exponentiations in G and one evaluation of H_2) and $\epsilon' = \epsilon + Q_P(Q_P + Q_H)/q$.*

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 are to Π_{SK} or F_{SK} (except on α^*)
- can distinguish between the two cases with advantage ϵ'

then we can build C that breaks (t, ϵ)-DDH assumption for $t \approx t'$ (plus the time for $\Theta(Q_H + Q_P)$ exponentiations in G and one evaluation of H_2) and $\epsilon = \epsilon' - Q_P(Q_P + Q_H)/q$.

Because F_{SK} is computable, in our case, from Π_{SK}, 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 Π_{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 either the correct VRF output 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 chooses a random $\rho_i \in \{1, \ldots, q\}$ and then 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 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 $\Pi_{|SK}$ (note that $q_i \neq \alpha$),
  - C makes a query to $H_1(q_i)$ 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 $s \in [q]$ 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 Π. The difference in how these distributions are obtained is simply that Π 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 Π; 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 $[q]$ and g is a generator), the chances that a single query to Π causes an abort are $(Q_H + Q_P)/q$, and the chances that any of the queries to Π causes an abort are $Q_P(Q_H + Q_P)/q$. Thus, the advantage of C is at least $\epsilon' - Q_P(Q_P + Q_H)/q$. □

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

Claim B.5. *Under the (t, ϵ)-DDH assumption, for any $Q_H \geq 1, Q_P$, the VRF satisfies $(t', Q_H, Q_P, \epsilon')$ pseudorandomness for output distribution $H_2(U_G)$, for $t' \approx t$ (minus the time for $\Theta(Q_H + Q_P)$ exponentiations in G and one evaluation of H_2) and $\epsilon' = 4\epsilon Q_P + Q_P(Q_P + Q_H)/q$.*

Proof. We explain the proof by showing the differences from the previous proof. 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 $\rho_i \in \{1, \ldots, q\}$. C remembers the type of the query and the ρ_i value.

If D makes a query q_i to Π, 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 instead of $F_{SK}(\alpha^*)$, D receives a uniform response chosen independently of anything else from 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 Π 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 multiplied by Q_P is increasing for $qsig \geq 1$, and its value at $Q_P = 1$ is $1/4$.) We thus obtain the claimed result. □

B.2.3 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 [66], 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 [66, Proof of Theorem 4]).

Definition B.6. (Trusted Collision Resistance) A VRF satisfies (Q_H, ϵ) trusted collision resistance if no adversary making Q_H random oracle queries, can, given an honestly generated SK, output two values $\alpha_1 \neq \alpha_2$ such that $F_{SK}(\alpha_1) = F_{SK}(\alpha_2)$ with probability greater than ϵ.

Claim B.7. *If every output of H_2 has at most τ preimages in G, then our VRF satisfies $(Q_H, \tau Q_H^2/(2q))$-trusted collision resistance. Note that in our suggested instantiation of H_2, $\tau = 2$, so we have $(Q_H, (Q_H + 2)^2/q)$-trusted collision resistance*

Proof. Let α_1, α_2 be the output of the adversary. Without loss of generality, assume α_1 and α_2 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.

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 $[H_1(\alpha^i)]^{xf}$ takes on one of the τ values that collide with $[H_1(\alpha^j)]^{xf}$ after the application of H_2. Since $H_1(\alpha^i)$ is uniform in $G - \{1\}$, and raising to xf is a permutation of $G - \{1\}$, the chances of hitting one of those τ values is $\tau/(q - 1)$. Applying the union bound over at most $(Q_H + 2)(Q_H + 1)/2$ pairs of distinct queries to H_1, we get that a successful output α_1, α_2 exists among queries to H_1 with probability at most $\tau(Q_H + 2)(Q_H + 1)/(2q - 2) < \tau(Q_H + 2)^2/(2q)$ (assuming the latter fraction is less than 1 — but the theorem statement is trivially true otherwise). □

Collision resistance without trusting the key. Similarly to the case with uniqueness, our VRF can be modified the same way 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.

C SECURITY OF RSA-BASED VRF

In [48] the authors provided an explicit proof only for the selective pseudorandomness of the RSA-based VRF in Figure 1 (see [48, Lemma III.2]), but not for its trusted uniqueness or for its collision resistance. These proofs are straightforward, but we provide them for completeness.

CLAIM C.1. *The RSA-based VRF of [48] satisfies trusted uniqueness as per [66, Definition 10]).*

PROOF. The claim that for every α there exist β, π such that $\mathsf{Ver}_{PK}(\alpha, \beta, \pi) = 1$ follows by inspection since for every α it is true that $\mathsf{Ver}_{PK}(\alpha, \mathit{Prove}_{SK}(\alpha)) = 1$.

Let A be an adversary such that $A(PK, SK) \to (\alpha, \beta_1, \pi_1)$ and $\mathit{Prove}_{SK}(\alpha) \to (\beta_2, \pi_2)$ and $\beta_1 \neq \beta_2$, where $(PK, SK) \leftarrow \mathit{Setup}(1^\kappa)$. Since $\beta_1 \neq \beta_2$ it follows that $\pi_1 \neq \pi_2$ as $\beta_i = H(\pi_i)$ for $i = 1, 2$ and $H(\cdot)$ implements a deterministic function. For the same reason, the value of $MGF(\alpha)$ is fully determined by α. Since PK, SK are valid RSA keys, the function $f(x) = x^e$ is a bijection in $\mathbb{Z}_N$ (where e is the RSA public exponent) and therefore $\pi_1^e \neq MGF(\alpha) = \pi_2^e$. Due to this, the probability that Ver_{PK} will accept for proof π_1 and value β_1 for input α is 0. □

CLAIM C.2. *The RSA-based VRF of [48] for H with output size ℓ (assuming ℓ is less than the length of the RSA modulus) satisfies $(Q_H, Q_H^2/2^{\ell+1})$-trusted collision resistance per definition B.6.*

PROOF. Indeed, for a collision to occur, either $H(\pi_1)$ should equal $H(\pi_2)$ for some $\pi_1 \neq \pi_2$, or $MGF(\alpha_1)$ should equal $MGF(\alpha_2)$ for $\alpha_1 \neq \alpha_2$. (Because trusted key generation ensures that raising to the power d is a permutation.) Let Q'_H be the number of queries to H and Q''_H be the number of queries to MGF. Let k be the output size of the MGF. The probability of collision, by the union bound, is at most $Q'^2_H/(2 \cdot 2^\ell) + Q''^2_H/(2 \cdot 2^k) \leq Q_H^2/2^{\ell+1}$ because $k \leq \ell$. □