NSEC5 from Elliptic Curves

Provably Preventing DNSSEC Zone Enumeration with Shorter Responses

Sharon Goldberg* Moni Naor† Dimitrios Papadopoulos* Leonid Reyzin*
* Boston University † Weizmann Institute

February 5, 2016

While DNSSEC securely provides authenticity and integrity to the domain name system (DNS), it also creates a new security vulnerability called *zone enumeration* [26, 10, 7, 41] that allows an adversary that asks a small number of targeted DNS queries to learn the IP addresses of all domain names in a zone. An enumerated zone can be used as "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" [26] (*e.g.,* per EU data protection laws, see [33],[2, pg. 37]), or to create a toehold for more complex attacks. As the Internet of things becomes increasingly ubiquitous, it also becomes increasingly important to keep the names and addresses of these 'things' (*e.g.,* thermostats, fridges, baby monitors) away from remote attackers.

In [19] we solved DNSSEC's zone enumeration problem by introducing NSEC5, a cryptographic construction based on RSA digital signatures. NSEC5 provides *authenticated denial of existence*, *i.e.,* it is used to answer DNS queries (*e.g.,* "What is the IP address for `aWa2j3.example.com`?") that do not have a response (*e.g.,* "`aWa2j3.example.com` is a non-existent domain.") RSA-based NSEC5 was recently submitted for specification in an Internet draft [39], and a working implementation of a nameserver that supports RSA-based NSEC5 is also available [1].

However, recent years have seen the DNSSEC community aiming to replace RSA with elliptic curve cryptography (EC), in order to shorten the length of DNSSEC responses [22, 38, 35]. Therefore, in this paper we present a new variant of NSEC5 that uses elliptic curve cryptography (ECC) to produce shorter NSEC5 responses. If a zone is signed with ECDSA at the 128-bit security level and also uses our new ECC-based NSEC5 scheme, its denial-of-existence responses (response code NXDOMAIN) will be about 2 times shorter than that a zone signed with 2048-bit RSA and RSA-based NSEC5 [19]. Moreover, our ECC-based NSEC5 has responses lengths that are comparable to NSEC3 [26], DNSSEC's current authenticated-denial-of-existence mechanism that is vulnerable to zone enumeration via offline dictionary attacks [41, 10]. In fact, if a zone signed with ECDSA at the 128-bit security level also uses our new ECC-based NSEC5 scheme with SHA-256 for hashing, it will have responses that are *shorter* than a zone using NSEC3 with 1024-bit RSA and SHA1 (for an 80-bit security level), which is today's dominant deployment configuration [36, 20].

Organization. Sections 1-2 start with background on the issues that lead to the development of NSEC5. We present our ECC-based NSEC5 construction in Sections 3-4; Section 3 explains how NSEC5 can be constructed from Verifiable Random Functions (VRF) [30, 27], and Section 4 presents our new ECC-based VRF for NSEC5. We estimate the response lengths for our NSEC5 scheme in Section 5. We conclude in Section 6 by precisely measuring response lengths for NSEC5 (as specified in its Internet draft [39]) using the NSEC5-ready implementation of the KNOT DNS nameserver [1] and empirical data from the `paypal.com` zone. Formal cryptographic proofs for the security of our scheme are in Appendix A.

1 Does length really matter?

The length of a DNSSEC response might, at first glance, seem like a foolish parameter to optimize. After all, when a DNS resolver sends a DNS query ("What is the IP address for `www.example.com`?") the nameserver produces a DNS response ("155.41.24.251") that is generally short enough to fit into a single IP packet. However, DNSSEC naturally amplifies the length of DNS response by including cryptographic digital signature along with information about cryptographic keys and algorithms (via the DNSSEC RRSIG record).

When amplification means that the DNSSEC response no longer fits in a single IP packet, several undesirable outcomes can occur. The response can be sent over UDP and fragmented across multiple IP fragments, and thus risks being dropped by a middlebox that blocks IP fragments [34, 37] or being subject to an IP fragmentation attack [20]. Alternatively, the resolver can be forced to resend the query over TCP [29], which incurs both a high performance cost (due to roundtrips needed to establish a TCP connection) and the risk that the connection will be dropped (because some middleboxes are configured to block DNS over TCP) [34].

Worse yet, DNSSEC length amplification can also be amplify DDoS attacks [16]. In an amplified DDoS attack, a botnet sends nameservers many short 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. Longer DNSSEC responses increase the volume of traffic that arrives at the victim. Fortunately, recent measurements [37, Fig. 10] found that DNSSEC's existing authenticated denial of existence records (*i.e.,* NSEC and NSEC3 records) are *not* the worst offenders in terms of DNSSEC length amplification. Our ECC-based NSEC5 has response lengths comparable to the NSEC3 responses that are already part of DNSSEC. (See Sections 5-6.)

2 Why today's DNSSEC trades off integrity and privacy.

We now review the issues that lead to the development of NSEC5 for DNSSEC.

Recall that 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 (155.41.24.251). *Nameservers* receive information from the zone owner, and respond to DNS queries for the zone made by *resolvers*. NSEC5 was developed to solve DNSSEC's zone enumeration problem, which arises when a DNS query ("What is the IP address for `aWa2j3.example.com`?") does not have a response ("`aWa2j3.example.com` is a non-existent domain.") Today, DNSSEC's methods for authenticated denial of existence reflect tradeoffs between privacy against zone enumeration and integrity of DNS information. We describe each method and its tradeoffs below:

NSEC (RFC 4034 [3]). 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 every consecutive pair of names; each pair of names is an NSEC record. 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 for the pair of existent names that are lexicographically before and after the non-existent name (`w.example.com` and `z.example.com`), as well as its associated DNSSEC signatures.

NSEC provides **strong integrity**—it not only protects against network attackers that intercept 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 highly vulnerable to zone enumeration attacks, where an adversary that makes a few *online* queries to the nameserver to collects all the NSEC records trivially learns all the names in the zone.

NSEC3 (RFC 5155 [26]). 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 signs every consecutive pair of hashes; each pair of hashes is an NSEC3 record. 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.

Like NSEC, NSEC3 uses precomputed responses, and thus also provides strong integrity. However, while hashing does make zone enumeration more difficult, it is still possible, as demonstrated by [10, 41] and acknowledged in RFC 5155 [26, Sec. 12.1.1]. To enumerate a zone that uses NSEC3, the adversary again makes a few *online* queries to the nameserver to collect all the NSEC3 records, and then uses an *offline* dictionary attacks to crack the hash values in the NSEC3 records, thus learning the names present in the zone. These offline attacks will only become faster and more efficient as technologies for hashing continue to improve (*e.g.,* GPUs [41], special-purpose ASICs).

NSEC3 White Lies (RFC 7129 [17]). NSEC3 White Lies was introduced to offer strong protection against zone enumeration. With NSEC3 White Lies, the nameserver holds the private ZSK and signs, online, the minimal possible pair of hash values that cover a queried name. 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 White Lies record only contains information about the queried name α, but not about any name present in the zone, it is not vulnerable to zone enumeration attacks. An adversary can only enumerate the zone by brute force—by sending an online query to the nameserver for each name that it suspects is in the zone. On the other hand, because the nameserver is required to hold the private ZSK and sign responses online, NSEC3 White Lies provides only **weak integrity**——it protects against network attackers that intercept DNSSEC responses, but is vulnerable to attacks by a malicious nameserver that uses the private ZSK to sign bogus DNSSEC responses. Weak integrity can be a problem when an organization's nameserver is run by a different organization,[1] or is compromised [40].

3 NSEC5: Strong integrity and strong privacy.

In [19], we introduced NSEC5 to provide both strong integrity and strong privacy against zone enumeration. 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)* [30].

3.1 Verifiable Random Functions (VRF).

A VRF [27] is essentially the public-key version of a keyed cryptographic hash. A VRF comes with a public-key pair (PK, SK), and only the holder of the private key SK can compute the hash, but anyone with public key PK can verify the hash. A VRF hashes an input α using the private key SK

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

[1]For example, the zone `umich.edu` has two authoritative nameservers run by the University of Michigan (`dns1.itd.umich.edu` and `dns2.itd.umich.edu`) and one run by the University of Wisconsin (`dns.cs.wisc.edu`) [32].

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 $\beta' \neq \beta$. See Section A for formal definitions.

It is sometimes the case that β can be computed directly from π by a simple operation, such as hashing. In this case communicating π alone to the recipient is enough. In particular, this is the case in our prior construction [19] and in the present paper.

3.2 NSEC5 from VRFs.

NSEC5 uses a VRF to provide authenticated denial of existence for DNSSEC [30, 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 whose keys are distinct from the 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 to distribute keys in DNSSEC.

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 strong 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.

Precomputing NSEC5 records. The trusted owner of the zone 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 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 NSEC5PROOF records. 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 then responds to the query with (1) an NSEC5PROOF record containing β and π, and (2) the precomputed NSEC5 record (and the associated DNSSEC signatures) for the pair of hashes lexicographically before and after β. (Notice that NSEC3 does not need a 'PROOF' record, since the resolver can hash α on its own; but it is exactly this public nature of the hash function in NSEC3 that leads to off-line zone enumeration attacks.)

Our VRF has the additional property that β is publicly computable from the proof π, so we will not need to include β in the NSEC5PROOF record.

Validating responses. The resolver validates the response by (1) using the public VRF key in the NSEC5KEY record to validate that proof π and hash output β from the NSEC5PROOF record correctly correspond to the query α, (2) checking that β falls between the two hash values in the signed NSEC5 record, and (3) using the public ZSK to validate the DNSSEC signatures on the NSEC5 record.

3.3 Properties of NSEC5.

NSEC5 requires online cryptographic computations; namely, for every query α that the nameserver receives, the nameserver computes the NSEC5PROOF record, on the fly, using the secret VRF key SK. Notice that 'NSEC3 White Lies' also requires online cryptographic computations; for every query α, the nameserver uses the secret ZSK to sign, on the fly, the minimally covering NSEC3 record for α (see Section 2). The fact that both of these solutions prevent zone enumeration is not a coincidence. Indeed, in [19] we 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, however, it that provides strong integrity (*i.e.,* integrity even when the nameserver is malicious or compromised), while 'NSEC3 White Lies' only provides weak integrity. We now sketch the security arguments for NSEC5; formal cryptographic proofs are in [30, Sec. 7].

Strong protection against zone enumeration. 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. To do this, 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.

Strong integrity. We now argue that, because the untrusted nameserver does not know the private ZSK, it cannot provide a false denial of existence (*i.e.,* to provide a valid response claiming that that name α does not exist in the zone, even though it really does). We emphasize that strong integrity is preserved even if a malicious nameserver, or any other adversary, knows the private VRF key SK. To see why, first observe that resolvers have the VRF public key PK and thus can check that the proof value π and hash output β in the NSEC5PROOF corresponds to their query α. Thus, by trusted uniqueness, no one, even given the knowledge of SK, can fool the resolvers into accepting a different β. Thus, since β cannot be forged, the only avenue of attack is to forge an NSEC5 record. (The forged NSEC5 record would contain a pair of hashes lexicographically before and after β; no true NSEC5 record of this form can exist, because β corresponds to a name α that exists in the zone) But forging an NSEC5 record requires forging a signature with the private ZSK, which the malicious nameserver (or other adversary) does not have.[2]

[2]Notice that the trusted uniqueness property of the VRF is crucial for providing strong integrity. This is why we cannot replace the RSA signature in our original NSEC5 construction from [19] with an ECDSA signature; RSA signatures are unique given the public key PK but ECDSA signatures are not. With randomized ECDSA signatures,

In fact, the argument for the strong integrity of NSEC3 is identical to the one we have just outlined. This is because if the private VRF key SK used with NSEC5 is made public, NSEC5 is essentially the same as NSEC3: the adversary can hash queries on its own, but cannot forge NSEC* records. Therefore, even if the private NSEC5KEY is leaked to an adversary, all that happens is that the security of NSEC5 downgrades to that of NSEC3.

4 A New Elliptic Curve VRF for NSEC5

We introduce a new VRF that operates in a cyclic group G of prime order with generator g and is secure in the random oracle model. Our VRF adapts the Chaum-Pederson (CP) protocol from CRYPTO'92 [14], for proving that two cyclic group elements g^x and h^x have the same discrete logarithm x base g and h, respectively. The security of this scheme, proven in the random oracle model, rests on the *computational Diffie-Hellman (CDH)* assumption, which roughly says that it is hard to compute h^x given the tuple (g, g^x, h). Our description adopts a multiplicative group notation, and all hash functions are modelled as random oracles.

4.1 Construction

Public parameters. Let q be a prime number, Z_q be the integers modulo q, $Z_q^* = Z_q - \{0\}$, and let G a cyclic group of prime order q with generator g. We assume that q, g and G are public parameters of our scheme. Let H_1 be a hash function (modeled as a random oracle) mapping arbitrary-length bitstrings onto the cyclic group G. (See Appendix B for a suggested instantiation of H_1.) Both hash functions H_2 and H_3 are (random oracles) instantiated with a standard cryptographic hash like SHA-256.

Keys. The secret VRF key $x \in Z_q$ is chosen uniformly at random. The public VRF key is g^x.

Hashing. Given the secret VRF key x and input α, compute the proof π as:
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 nonce $k \in Z_q$.
3. Compute $c = H_3(g, h, g^x, h^x, g^k, h^k)$.
4. Let $s = k - cx \bmod q$.

The proof is the group element γ and the two exponent values c, s. The VRF output $\beta = F_{SK}(\alpha)$ is computed by hashing γ with H_2 . Thus

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

Notice that anyone can compute the VRF hash output β given the proof π. (This is why we don't need to include β in the NSEC5PROOF record, per Section 3.2.)

Verifying. Given public key g^x , verify that proof π corresponds to the input α and output β as follows:
1. Given public key g^x, and exponent values c and s from the proof π, compute $u = (g^x)^c \cdot g^s$. Note that if everything is correct then $u = g^k$.

the signer uses a nonce as part of its signature computation, and so signatures are not unique given the ECDSA public key PK. Moreover, even deterministic ECDSA [31] fails to provide uniqueness given the ECDSA public key PK. This follows because with deterministic ECDSA, the signer derives the signing nonce from a keyed hash of the message it is signing, but the key k to this hash in *independent* of the ECDSA public key PK. Thus, PK does not allow the verifier to check that the signer used the correct nonce for each message it signed.

2. Given input α, hash it to obtain $h = H_1(\alpha)$. Make sure that $\gamma \in G$. Use h and the values (γ, c, s) from the proof to compute $v = (\gamma)^c \cdot h^s$. Note that if everything is correct then $v = h^k$.
3. Check that hashing all these values together gives us c from the proof. That is, given the values u and v that we just computed, the group element γ from the proof, the input α, the public key g^x and the public generator g, check that:

$$c = H_3(g, H_1(\alpha), g^x, \gamma, u, v) \qquad \textit{i.e.,} \qquad c = H_3(g,\ H_1(\alpha),\ g^x,\ \gamma,\ (g^x)^c \cdot g^s,\ (\gamma)^c \cdot [H_1(\alpha)]^s)$$

Finally, given γ from the proof π, check that $\beta = H_2(\gamma)$.

4.2 Instantiation with elliptic curves.

Our VRF can be instantiated over any group where the computational Diffie-Hellman (CDH) problem is hard, including the elliptic curves currently standardized in DNSSEC (NIST P-256 [23, Sec. 3]), and the Curve25519 [25] which has recently been proposed for use with DNSSEC [24, 35]. Each of these curve achieves an approximately 128-bit security level, roughly as strong as 3072-bit RSA [9, 23]. Both of these curves operate in finite field F_q, where q is a 256-bit prime, therefore the exponents in our construction are of size 256 bits. Using point compression, each elliptic curve point can be represented with only 256+1 bits, so the proof π will be $p = 256 + 1 + 2 * 256 = 769$ bits long.

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 $2 * 256 + 2 * 256 = 1024$ 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 [38]. However, as Bernstein [8] argues: "a patent cannot cover compression mechanisms [appearing in the paper by Miller in 1986 [28] that was] published seven years before the patent was filed." Indeed, current Internet drafts for Curve25519 for DNSSEC signatures also use point compression [24].

4.3 Security properties.

Formal proofs that this VRF satisfies selective pseudorandomness and trusted uniqueness are in Section A. We sketch the proofs here.

Trusted 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

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

Now since $\gamma_1 \neq h^x$ (since β_1 is not the correct output value), the denominator is not zero, and there is exactly one c_1 modulo q that satisfies equation (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 (1) is negligible. Thus, we have arrived at our contradiction.

Pseudorandomness. This follows from the CDH 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 CDH assumption, its hard for this adversary to compute h^x even given (g, g^x, h). Since the adversary does not know h^x, the random oracle output $\beta = H_2(h^x)$ looks entirely random to him, and pseudorandomness follows.

5 Estimating the length of responses.

We now perform some back-of-the-envelope length estimates for NSEC5 with our elliptic curve VRF, and compare it to NSEC3 and our RSA-based NSEC5 construction from [19]. (In [19], we constructed NSEC5 from a VRF whose proof value π was a RSA signature, and the VRF hash output was $\beta = H(\pi)$ where H is standard cryptographic hash function.[3]) These estimates only take into account the cryptographic material (*i.e.,* hashes, signatures, proofs) associated with each response. We assume that all three mechanisms use cryptographic hash functions H producing outputs of length ℓ and DNSSEC signatures (*i.e.,* RRSIG records) producing outputs of length σ. NSEC5 proofs have length p.

Length estimates. NSEC5 records and NSEC3 records are almost identical in structure; each contains two hash value and is signed with an RRSIG record, and thus incurs a length of $2\ell + \sigma$. NSEC3 deals with wildcards and closest enclosers by returning (in the worst case) *three* NSEC3 records for a given query [26]. It follows that the length of an NSEC3 response is, in the worst case

$$|\text{nsec3}| = 3(2\ell + \sigma) = 6\ell + 3\sigma \tag{2}$$

However, in [18], Gieben and Mekking observed that wildcards and closest enclosers could be dealt with using only *two* NSEC3 records. While this optimization was not standardized as part of NSEC3, we can apply it to our setting because NSEC5 records are almost identical to NSEC3

[3]Our original NSEC5 construction [19] did not describe NSEC5 in terms of VRFs. However, it did actually use the following VRF, which is based on RSA in the random oracle model. To hash input α using the private RSA key SK, start by computing the proof value

$$\pi = RSASIG_{SK}\left(MGF(\alpha)\right)$$

and then compute the hash value β as

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

H is a standard cryptographic hash function (*e.g.,* SHA-256) while MGF is an industry-standard cryptographic hash that produces outputs one bit shorter than the RSA modulus [4, Sec. 10.2]. To verify that β is indeed the hash of α, (1) use the public RSA key PK to verify that π is a valid RSA signature on $H_1(\alpha)$, and (2) verify that $H(\pi) = \beta$. Notice that trusted uniqueness follows because, given the public key PK, then $RSASIG_{SK}(MGF(\cdot))$ can produce exactly one proof value π for every input value α. Like our ECC VRF, this RSA-based VRF also has the property that the hash value β is publicly computable from the proof π; as such, in [19] we also did not need to include β in the NSEC5PROOF record.

Hash Algorithm	SHA1, $\ell = 160$ bits			SHA-256, $\ell = 256$ bits		
	RSA		ECDSA	RSA		ECDSA
Signature Algorithm	$\sigma = 1024$	$\sigma = 2048$	$\sigma = 512$	$\sigma = 1024$	$\sigma = 2048$	$\sigma = 512$
NSEC3	4032	7104	2496	4608	7680	3072
NSEC5 RSA, $p = 1024$	4736	6784	3712	5120	7168	4096
NSEC5 RSA, $p = 2048$	6784	8832	5760	7168	9216	6144
NSEC5 EC, $p = 769$	4226	6274	3202	4610	6658	3586

Table 1: Length estimates for different mechanisms for authenticated denial of existence. All numbers shown in bits and computed per equations (2), (3). As discussed, SHA1 is considered to be end-of-life [5] and thus the left side of this table corresponds to the, insecure but still widely used, 80-bit security level, while the right side of the table reflects the use of the more modern SHA-256 hash function for a 128-bit security level. Each column represents a different signature algorithm for RRSIG records.

records. Therefore with NSEC5 we require only *two* NSEC5 records along with their associated RRSIG and NSEC5PROOF records in response to every query. Every NSEC5PROOF is unsigned and contains a proof value π of length p, where our elliptic curve VRF has $p = 769$ and our RSA VRF has p equal to the length of an RSA signature. Thus, the length of an NSEC5 response is, in the worst case

$$|\text{nsec5}| = 2(2\ell + \sigma + p) = 4\ell + 2\sigma + 2p \tag{3}$$

Cryptographic algorithms and key lengths. Modern key-management recommendations mandate a security level of at least 112 bits for cryptographic applications. In practice this corresponds to signatures with 2048-bit RSA ($\sigma = 2048$ bits) or ECDSA over 256-bit curves ($\sigma = 512$ bits) where SHA-256 is the hash function ($\ell = 256$ bits) [6]. However, the majority of domains that employ DNSSEC still use 1024-bit RSA for signing [21, 36], to ensure shorter responses; this corresponds to an 80-bit security level that is widely considered outdated and insecure [6]. Moreover, NSEC3 currently only supports SHA1 as the hashing function [26], whereas NSEC5 was proposed with SHA-256 as the default option [39] in accordance with modern security standards. For completeness we include in our analysis all possible combinations of schemes and hash functions, *i.e.,* NSEC3 and NSEC5 with both SHA1 and SHA-256. Table 1 instantiates equations (2) and (3) with these values of signature length σ and hash output length ℓ and NSEC5 proof length p.

Observations. From Table 1, we see that the NSEC5 responses are shortest when elliptic curves are used both for the signature (*i.e.,* ECDSA) and for the NSEC5 proof (per Section 4); when SHA-256 is used as the hash function, the cryptographic material in the NSEC5 responses is about ≈ 3586 bits long. We can immediately see the improvement if we compare the elliptic curve approach with RSA; NSEC5 with SHA-256 and 2048-bit RSA signatures and proofs has ≈ 9216 bits of cryptographic material, so ECC-based NSEC5 responses are 2.6 times shorter!

Moreover, NSEC5 with elliptic curves for both the signature and the proof provides responses (≈ 3586 bits) that are not much longer than NSEC3 with elliptic curve signatures and SHA-256 (≈ 3072 bits). In fact, NSEC5 with elliptic curves at the 128-bit security level (≈ 3586 bits) produces responses that are *shorter* than NSEC3 at a lower security level (*i.e.,* ≈ 4032 bits for NSEC3 with 1024-bit RSA and SHA1); this last comparison is especially significant since many DNSSEC zones today are still signed using 1024-bit RSA [36, 21].

6 Measuring response lengths with an NSEC5-ready nameserver

We now validate the observations made in Section 5 by precisely measuring response lengths using the NSEC5-ready implementation of the KNOT DNS nameserver from [1] and empirical data from the `paypal.com` zone. We determine the length of responses for NSEC, NSEC3 and NSEC5 with signing algorithms that have a security level of at least 112-bits. That is, for producing RRSIG records we consider both 2048-bit RSA ($\sigma = 2048$) and ECDSA over the NIST P-256 curve ($\sigma = 512$). For each scheme we used the only standardized hash function, *i.e.,* SHA1 ($\ell = 160$) and SHA-256 ($\ell = 256$) for NSEC3 and NSEC5 respectively.

6.1 Measurement methodology.

We chose `paypal.com` because it is the only Alexa-100 domain that deploys DNSSEC; this zone uses NSEC and so it could be trivially enumerated with a small number of online queries. Therefore, in 10/2015, we collected all the NSEC records from `paypal.com` and we ended up with a zone of a little more than 1000 records.

To sign the zone, we used the NSEC5-ready version of the KNOT DNS nameserver from [1], which implements our RSA-based NSEC5 scheme from [19] as specified in the IETF Internet draft [39]. We used this nameserver to re-sign the zone using (1) NSEC with 2048-bit RSA signatures, (2) NSEC with 512-bit ECDSA signatures, (3) NSEC3 with 512-bit ECDSA signatures, (2) NSEC3 with 2048-bit RSA signatures, (5) NSEC5 with 2048-bit RSA signatures and proofs, (6) NSEC5 with 2048-bit RSA proofs and 512-bit ECDSA signature (a *hybrid* EC/RSA scheme). For each instance of the signed zone, we collected denial-of-existence responses (response code NXDOMAIN[4]), by issuing DNS queries with a random five-character string prepended to each actual record name from the zone. We report the average and standard deviation of the length of these responses in Figure 1. Note that this measurement is very accurate, as it includes all the appropriate records and fields in the response packet, rather than just the cryptographic material as discussed in Section 5.

Finally, because the nameserver implementation [1] does not yet support NSEC5 with ECC proofs, we calculated the response size for (7) NSEC5 with ECC proofs and ECDSA signatures by taking a delta from the hybrid EC/RSA NSEC5 scheme; this delta is concretely measured as $2048 - 769 = 1279$ bits per proof. (We emphasize that this is still a very accurate estimation of the final response size, as the *only* difference in packet formats between ECC-based NSEC5 and RSA-based NSEC5 is the owner name hash field of the NSEC5PROOF record, as specified in [39].)

6.2 Observations.

MTU. In all cases, transition from RSA to ECDSA reduces the response size significantly. All schemes produce packets larger than 512 bytes (which is the default UDP size limit for DNS when EDNS0 is not used) and most of them are within the Ethernet Maximum Transmission Unit (MTU) which is 1500 bytes.

Variance in response length. The error bars in Figure 1 correspond to one standard deviation. Notice the NSEC5 has much smaller variance than NSEC or NSEC3. This follows because with NSEC, some queries are handled with a single NSEC + RRSIG record (if the same record covers both the queried name and the wildcard expansion) while others are handled with two NSEC +

[4]We focused on NXDOMAIN responses because, out of all the "negative" response types (such as NODATA or empty non-terminals), they typically are the longest.

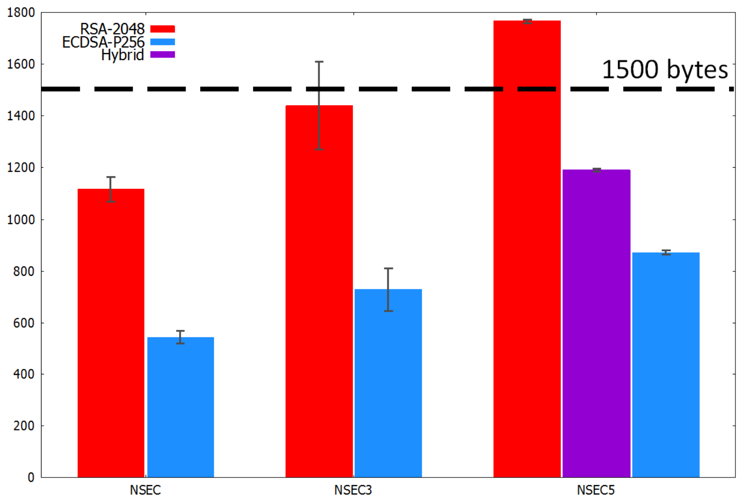

Figure 1: Average and standard deviation of NXDOMAIN response size in bytes.

RRSIG records. Meanwhile, with NSEC3, if the closest encloser happens to be an empty non-terminal, the response contains two NSEC3 + RRSIG records instead of three. On the other hand, NSEC5 responses *always* have two NSEC5 + RRSIG + NSEC5PROOF records, and the (small) variance is only due to differences in the record content.

RSA-based approaches. As can be seen in the figure, NSEC5 with RSA proofs and signatures can significantly increase response size. If 2048-bit RSA is used for NSEC3 or NSEC5 (middle and rightmost red bars in the plot) packets grow prohibitively large, above or close to the Ethernet MTU. This leads to a variety of undesirable outcomes including packet fragmentation (Section 1), and further justifies the recent push for adoption of elliptic curve cryptography for DNSSEC [38, 15, 22]).

ECC-based approaches. Leaving RSA behind, the current state-of-the-art NSEC3 with ECDSA has an average response length of 751 bytes (middle blue bar). Comparing this to the hybrid NSEC5 scheme with RSA proofs and ECDSA signatures (which was achievable prior to this paper), the average response length is 1190 bytes (middle purple bar). This still corresponds to a significant length increase of $\sim$ 58.4%. Meanwhile if we use a full ECC approach with NSEC5 (ECC proofs and ECDSA signatures), the average response size is 870 bytes (rightmost blue bar), corresponding to a mere $\sim$ 15.9% increase over the best possible with NSEC3 variant. Note that these numbers are achieved while using SHA1 for hashing at NSEC3 and SHA-256 for NSEC5. In case NSEC5 is standardized for use with SHA1, then the gap between the two drops to just 5.1%.

ECC-based NSEC5 is shorter than a NSEC3 with 1024-bit RSA. Quite surprisingly, our NSEC5 scheme not only eliminates zone enumeration and achieves better security level (128-bit vs. 80-bit security), but also produces *shorter* responses than today's dominant deployment configuration [36]. Specifically, NSEC3 using 1024-bit RSA and SHA1 (for an 80-bit security level) produces response lengths of 1038 bytes on average. This is approximately 20% *longer* than what we achieve with NSEC5 with EC proofs and ECDSA signatures at the 128-bit security level!

Acknowledgements

We are grateful to Jan Včelák for useful discussions and for encouraging us to obtain these results, and to Duane Wessels and Tomofumi Okubo for their valuable feedback. We also thank innumerable other DNS practitioners at DNS OARC Fall'14, IETF'92 and IETF'93 who asked if we could build NSEC5 from elliptic curves. This work was supported by NSF grants 1012798 and 1012910 and a gift from Verisign Labs.

References

[1] NSEC5-ready nameserver based on the KNOT DNS software. `https://github.com/dipapado/nsec5-implementation`, ((Accessed 1/15/2016)).

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

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

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

[5] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for Key Management Part 1: General (Revised). NIST Special Publication 800-57, March 2007.

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

[7] Jason Bau and John C. Mitchell. A security evaluation of dnssec with nsec3. In *NDSS*, 2010.

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

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

[10] Daniel J. Bernstein. Nsec3 walker. `http://dnscurve.org/nsec3walker.html`, 2011.

[11] Dan Boneh, Ben Lynn, and Hovav Shacham. 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*, pages 514–532, 2001.

[12] Colin Boyd, Paul Montague, and Khanh Nguyen. Elliptic curve based password authenticated key exchange protocols. In Vijay Varadharajan and Yi Mu, editors, *Information Security and Privacy*, volume 2119 of *Lecture Notes in Computer Science*, pages 487–501. Springer Berlin Heidelberg, 2001.

[13] Melissa Chase and Anna Lysyanskaya. Simulatable vrfs with applications to multi-theorem NIZK. In *CRYPTO'07*, pages 303–322, 2007.

[14] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in CryptologyCRYPTO92*, pages 89–105. Springer, 1992.

[15] CloudFlare. DNSSEC Complexities and Considerations. `https://www.cloudflare.com/dnssec/dnssec-complexities-and-considerations/`.

[16] Steve Gibson. Distributed Reflection Denial of Service (DrDoS) Attacks. Technical report, Gibson Research Corporation, 2002.

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

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

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

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

[21] Amir Herzberg and Haya Shulman. Negotiating dnssec algorithms over legacy proxies. In *Proceedings of the 13th International Conference on Cryptology and Network Security - Volume 8813*, pages 111–126, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[22] Amir Herzberg and Haya Shulman. Cipher-suite negotiation for dnssec: Hop-by-hop or end-to-end? *Internet Computing, IEEE*, 19(1):80–84, 2015.

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

[24] S. Josefsson and N. Moeller. draft-josefsson-eddsa-ed25519-03:EdDSA and Ed25519, 2015. `https://tools.ietf.org/html/draft-josefsson-eddsa-ed25519-03`.

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

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

[27] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.

[28] Victor Miller. Use of elliptic curves in cryptography. In *Advances in CryptologyCRYPTO85 Proceedings*, pages 417–426. Springer, 1986.

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

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

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

[32] Venugopalan Ramasubramanian and Emin Gün Sirer. Perils of transitive trust in the domain name system. In *ACM IMC*, 2005.

[33] Marcos Sanz. Dnssec and the zone enumeration. European Internet Forum: `http://www.denic.de/fileadmin/public/events/DNSSEC_testbed/zone-enumeration.pdf`, October 2004.

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

[35] O. Sury and R. Edmonds. *draft-sury-dnskey-ed25519: Ed25519 for DNSSEC*. Internet Engineering Task Force (IETF), 2015. `https://tools.ietf.org/html/draft-sury-dnskey-ed25519-03`.

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

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

[38] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. Making the case for elliptic curves in dnssec. *ACM SIGCOMM Computer Communication Review*, 45(5):13–19, 2015.

[39] J. Vcelak and S. Goldberg. draft-vcelak-nsec5-01:NSEC5, DNSSEC Authenticated Denial of Existence, 2015. `http://tools.ietf.org/html/draft-vcelak-nsec5-01`.

[40] Cory von Wallenstein. DNS Security: Three Ways That Hijacks Can Happen. Dyn Blog `http://dyn.com/blog/dns-101-explaining-how-hijacks-can-happen/`, August 2013.

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

A Security Proofs

We prove that the construction in Section 4 is a secure VRF. It suffices to prove two properties: Trusted Uniqueness (see [30, Definition 10]) and Selective Pseudorandomness (see [30, Definition 11]). Sufficiency of these two properties for constructing NSEC5 follows from [30, Theorem 4]. We discuss each property in turn.

We model all hash functions (*i.e.,* H_1, H_2, H_3) as random oracles. We use the notation *Prove* to denote operation that on input α uses the secret VRF key SK to produce a VRF proof value π and VRF hash value β, *i.e., Prove*$_{SK}(\alpha) \to (\beta, \pi)$. The notation $\mathsf{Ver}_{PK}(\alpha, \beta, \pi) = 1$ denotes the algorithm that uses the public key PK to verify that proof π and hash output β are valid for input α.

A.1 Trusted Uniqueness.

Following tradition of the VRF literature, Naor and Ziv [30, Definition 10]) define Trusted 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, our constructions satisfies it only computationally: more than one hash output y may exist, but only one valid β—the one produced by *Prove*—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 [13] mention that such a relaxation can be defined. We therefore define it here.

Definition A.1 (Computational Trusted Uniqueness). *For all probabilistic polynomial-time adversaries A, for a random validly chosen key pair $(PK, SK) \leftarrow Setup(1^\kappa)$,*

$$\Pr[A(PK, SK) \to (\alpha, \beta_1, \pi_1) \text{ and } Prove_{SK}(\alpha) \to (\beta_2, \pi_2) \text{ s.t. } \beta_1 \neq \beta_2 \text{ and } \mathsf{Ver}_{PK}(\alpha, \beta_1, \pi_1) = 1]$$

is negligible in κ, where κ is the security parameter.

(The second part of the definition—namely, that for every PK, for an overwhelming fraction of the domain of α, there exists β and π such that $\mathsf{Ver}_{PK}(\alpha, \beta, \pi) = 1$—remains unchanged.)

We now prove that our VRF satisfies this property unconditionally (based on the randomness of the oracle H_3 and not on any computational assumptions).

Claim A.2. *Our VRF satisfies computational trusted uniqueness of Definition A.1.*

Proof. Suppose there is an adversary A that violates uniqueness with probability ϵ. That is, on input g, x, the adversary A makes Q queries to the H_3 oracle and wins by outputting (α, β_1, π_1) s.t. $\beta_1 \neq \beta_2$ and $\mathsf{Ver}(\alpha, \beta_1, \pi_1) = 1$ with probability ϵ. We will show that $\epsilon \leq (Q + 1)/\min(q/2, \rho)$, where q is the order of the group G and $\rho = |\text{range}(H_3)|$. Note that this negligible in κ, because the group size $|G|$ and the output range of H_3 are exponential in κ, while Q is bounded by the adversary's polynomial running time.

The proof π_1 contains γ_1 such that $\beta_1 = H_2(\gamma_1)$; similarly, π_2 contains γ_2 such that $\beta_2 = H_2(\gamma_2)$. Since $\beta_1 \neq \beta_2$, we have $\gamma_1 \neq \gamma_2$. Therefore, since $\gamma_2 = [H_1(\alpha)]^x$ (because that is what *Prove* produces), we have $\gamma_1 \neq [H_1(\alpha)]^x$.

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 computes $h = H_1(\alpha)$ and

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

Assume, for now, that $h \neq 1$, and thus (because G is of prime order), h is a generator of G. Then we can take the logarithm of the first equation base g and the logarithm of the second equation base h, because h and g are generators of G. Solving these for s we get

$$\log_g u - cx \equiv s \pmod{q}$$
$$\log_h v - c \log_h \gamma_1 \equiv s \pmod{q}$$

which implies that

$$c \equiv \frac{\log_g u - \log_h v}{x - \log_h \gamma_1} \pmod{q} \tag{4}$$

Since $\gamma_1 \neq h^x$, 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.

If $h = 1$, then $\gamma_2 = h^x = 1$, and so $\gamma_1 \neq 1$. Thus, γ_1 is a generator of G, and so again c modulo q is uniquely determined given g, PK, h, γ_1, u, and v: namely

$$c = \log_{\gamma_1} v \,. \tag{5}$$

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 the equations (4) and (5), and thus the correct c is uniquely defined at the time the query is made.

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 Z_q, and the correct c is in that range, then this probability is at most $1/\rho$. If ρ is an exact multiple of q, then this probability is at most $1/q$. Finally, if ρ not an exact multiple of q but is greater than q, then some values modulo q are more likely than others, but none will have probability greater than $\lceil \rho/q \rceil/\rho < 2/q$.

Assume the adversary outputs β_1, π_1 and then the verification algorithm is run. This causes a total of $Q + 1$ queries to H_3 (Q 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 $(Q + 1)/\min(q/2, \rho)$.

The second part of trusted uniqueness (namely, that verifiable y and π exists for most α) is true by inspection: β and π exists for every α, because the algorithm *Prove* can be used to produce them. □

Remark. Since our computational trusted uniqueness property is slightly weaker than the unconditional trusted uniqueness of [30, Definition 10], the proof of the soundness property in [30, Theorem 4] of Naor-Ziv needs a slight change, as follows. Recall that the proof 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 [30]) and proof π for the VRF as part of its soundness-violating output on an input α (called x in [30]). Computational trusted soundness ensure that this happens except with negligible (i.e., $(Q + 1)/\min(q/2, \rho)$) probability. Thus, the success probability of the reduction reduces from ϵ to $\epsilon - (Q + 1)/\min(q/2, \rho)$.

A.2 Selective Pseudorandomness.

We will state selective pseudorandomness in terms of concrete, rather than asymptotic, security. This type of security is more relevant to practice, and, in our case, avoids the formalism of defining how G is produced as a function of the security parameter and how the adversary gets a description of G. Instead, we work for fixed G.

Selective Pseudorandomness of our VRF depends on the following assumption about the group G and generator g, known as the (t, ϵ)-*CDH Assumption*: for any adversary C whose description size and running time are bounded by t, the probability (over a random choice of $h \in G$ and $x \in Z_q$) that $C(g, g^x, h)$ outputs h^x is at most ϵ.

Claim A.3. *Under the (t, ϵ)-CDH assumption, our VRF satisfies Selective Pseudorandomness as defined in [30, Definition 11]: namely, no adversary F with running time (plus description size) t' that makes at most Q queries to Prove and the random oracles (of which at most Q_P are to Prove) can break pseudorandmomness with advantage more than ϵ', where $t' \approx t$ (minus the time for $\Theta(Q)$ exponentiations in G) and $\epsilon' = Q(\epsilon + (Q_P + 1)/q)$.*

Proof. We need to show the following: if

- F chooses α,
- then receives an honestly generated $PK = g^x$ and is allowed Q total oracle queries, of which at most Q_P queries are to $\mathit{Prove}_{G,g,x}$ (except on α) and the rest are to random functions H_1, H_2, H_3,
- and then distinguishes $H_2([H_1(\alpha)]^x)$ from a random bit string of length n with advantage ϵ',

then we can build C that breaks (t, ϵ)-CDH assumption for $t \approx t'$ (plus the time for $\Theta(Q)$ exponentiations in G) and $\epsilon = \epsilon'/Q - (Q_P + 1)/q$.

Note that if, during this attack, no query to H_2 on input $[H_1(\alpha)]^x$ is made by F or its *Prove* oracle, then $H_2([H_1(\alpha)]^x)$ is distributed uniformly at random, independently of the view of F, and is thus perfectly indistinguishable from random. Therefore, for F to have advantage ϵ', H_2 must be queried on $[H_1(\alpha)]^x$ by either F or its *Prove* oracle with probability at least ϵ'.

Since F never queries *Prove* on α, the only way *Prove* will query H_2 on $[H_1(\alpha)]^x$ is if $H_1(\alpha) = H_1(\alpha_i)$ for some query α_i that F makes to *Prove*. Recall that α is fixed by F ahead of time; thus, the probability that F is able to find α_i with $H_1(\alpha) = H_1(\alpha_i)$ is at most Q/q, by the union bound over the total number of queries to H_1. Therefore, $[H_1(\alpha)]^x$ must be queried to H_2 with probability at least $\epsilon' - Q/q$. We will use this fact to break the CDH assumption, as follows.

Given (g, g^x, h), C gets α from F, sets (g, g^x) as the VRF public key PK and runs $F(PK)$, answering the queries of F as follows (note that all random oracle query responses are computed as below unless already defined):

- If F queries α to random oracle H_1, C returns h.
- If F queries any other α_i to H_1, C chooses a random $\rho_i \in [q]$ and then programs H_1 as

$$H_1(q_i) := g^{\rho_i}\,.$$

  (note that this response is distributed uniformly in G, just a like the honest H_1, because g is a generator of G).
- If F queries H_2 or H_3, C return a fresh random value in the appropriate range (note that these responses are distributed just like honest H_2 and H_3).
- If F makes a query q_i to *Prove* (note that $q_i \neq \alpha$),
  - C makes a query to $H_1(q_i)$ as described above to get ρ_i,
  - C makes a query to $H_2((g^x)^{\rho_i})$ as described above to get β,
  - C chooses a random values $s \in [q]$ and $c \in \text{range}(H_3)$ and then computes

$$u = g^s (g^x)^c$$

    and

$$v = [g^{\rho_i}]^s [(g^x)^{\rho_i}]^c\,.$$

    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 H_3 oracle to let

$$H_3(g, g^{\rho_i}, g^x, (g^x)^{\rho_i}, u, v) := c$$

    (note that 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 F is faithful and, as argued above, the H_2 query table will therefore contain $[H_1(\alpha)]^x = h^x$ with probability at least $\epsilon' - Q/q$. The probability that C aborts is simply the probability that $H_3(g, g^{\rho_i}, g^x, (g^x)^{\rho_i}, u, v)$ is already defined during the computation of the response to *Prove*; since at most Q 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 *Prove* causes an abort are Q/q, and the chances that any of the queries to *Prove* causes an abort are $Q_P Q/q$. Thus, the H_2 query table contains h^x with probability at least $\epsilon' - (Q_P + 1)Q/q$. C simply outputs a random element in this table; since there are at most Q of them, C succeeds with probability $\epsilon'/Q - (Q_P + 1)/q$. $\square$

B Hashing onto the curve.

Our VRF (Section 4) 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 [11] and, at a high level, it proceeds as follows. Assume an elliptic curve with corresponding equation $y^2 = x^3 + ax + b$ (in Weierstrass form). Given an integer α (the queried url 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 $(h, (h^3 + ax + b)^{1/2})$. 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 $(O \log^3(q))$, if the curve is defined over finite field F_q. 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 Claim A.3 $1/q$ gets subtracted from the success of C, and the running time for simulating queries to H_1 doubles).

As first shown in [12], 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.