NSEC5: Provably Preventing DNSSEC Zone Enumeration

Sharon Goldberg*, Moni Naor†, Dimitrios Papadopoulos*, Leonid Reyzin*, Sachin Vasant* and Asaf Ziv†
*Boston University, Department of Computer Science
Email: {goldbe,dipapado,reyzin,sachinv}@cs.bu.edu
†Weizmann Institute of Science, Department of Computer Science and Applied Mathematics
Email:{moni.naor,asaf.ziv}@weizmann.ac.il

***Abstract*—We use cryptographic techniques to study zone enumeration in DNSSEC. DNSSEC is designed to prevent attackers from tampering with domain name system (DNS) messages. The cryptographic machinery used in DNSSEC, however, also creates a new vulnerability, *zone enumeration*, enabling an adversary to use a small number of online DNSSEC queries combined with offline dictionary attacks to learn which domain names are present or absent in a DNS zone.**

We prove that the current DNSSEC standard, with NSEC and NSEC3 records, inherently suffers from zone enumeration: specifically, we show that security against (1) attackers that tamper with DNS messages and (2) privacy against zone enumeration cannot be satisfied simultaneously, unless the DNSSEC nameserver performs *online* public-key cryptographic operations.

We then propose a new construction that uses online public-key cryptography to solve the problem of DNSSEC zone enumeration. NSEC5 can be thought of as a variant of NSEC3, in which the unkeyed hash function is replaced with a deterministic RSA-based *keyed hashing* scheme. With NSEC5, a zone remains protected against network attackers and compromised nameservers even if the secret NSEC5-hashing key is compromised; leaking the NSEC5-hashing key only harms privacy against zone enumeration, effectively downgrading the security of NSEC5 back to that of the current DNSSEC standard (with NSEC3).

I. Introduction

DNSSEC was introduced in the late 1990s to protect the Domain Name System (DNS) from network attacks. With DNSSEC, the response to a DNS query is authenticated with a digital signature; in this way, the resolver that issues the DNS query ("What is the IP address for `www.example.com`?") can be certain that the response ("155.41.24.251") was sent by the authoritative nameserver, rather than an arbitrary attacker. The road to DNSSEC deployment has been rocky, and a variety of technical issues have forced the Internet community to rewrite the DNSSEC standard several times. One of the most interesting of these issues is the problem of *zone enumeration* [2], [9], [13]. Zone enumeration allows an adversary to learn the IP addresses of all hosts in a zone (including routers and other devices), creating a toehold from which it can launch more complex attacks. While a number of standards (RFC 4470 [47], RFC 5155 [30]) have tried to fix the zone enumeration problem, a complete solution to the problem has remained mysteriously elusive. In this paper, we use cryptographic lower bounds to explain why all previous techniques based on hashing failed to solve the problem. Our result shows that achieving privacy guarantees in this setting (while preserving the security properties of DNSSEC) necessitates the use of public-key cryptographic operations in the online phase of the protocol. Moreover, we present NSEC5, a new cryptographic construction that solves the problem of zone enumeration while remaining faithful to the operational realities of DNSSEC.

A. DNSSEC.

For the purpose of understanding the zone enumeration problem, we can partition the functionalities of DNSSEC into two distinct parts. The first is to provide an authenticated *positive* response to a DNS query. (For example, query: "What is the IP address for `www.example.com`?"; answer: "`www.example.com` is at 155.41.24.251.")

The second is to provide an authenticated *denial of existence*, when no response to the query is available. (For example, query: "What is the IP address for `aWa2j3.example.com`?"; answer: "`aWa2j3.example.com` is a non-existent domain.") DNSSEC deals with these functionalities in different ways.

For positive responses, the authoritative nameserver for the zone (*i.e.,* the nameserver that is authorized to answer DNS queries for domains ending in `example.com`) keeps a finite set R of signed resource records; each record contains a mapping from one domain name to its IP address(es) and is signed by the zone's secret keys. Importantly, these signatures need not be computed online in response to live DNS queries, but instead are precomputed ahead of time and stored at the nameserver. This has the twin advantages of (1) reducing the computational load at the nameserver, and (2) eliminating the need to trust the nameserver (since it need not store the zone signing key). This second advantage is especially important because most zones have more than one authoritative nameserver, and some nameservers might even be operated by entirely different organizations than the one that administers

First posted July 25, 2014. Last updated December 5, 2014.
This is the authors' full version of the paper that appeared at NDSS '15, 8-11 February 2015, San Diego, CA, USA. http://eprint.iacr.org/2014/582

the zone[1]. In what follows, we will use the term *primary nameserver* (or simply *primary*) to describe nameservers that are trusted, and *secondary nameservers* (or simply *secondary*) to describe those that are not.

B. The DNSSEC Zone Enumeration Problem

The zone enumeration problem becomes an issue when we consider DNSSEC negative responses. The trivial idea of responding to every query for a non-existent domain with the precomputed signed message "Non-existent domain" opens the system up to replay attacks. Another trivial idea of precomputing signed responses of the form "______ is a non-existent domain" also fails, since the number of possible queries that deserve such a response is infinite, making precomputation of signed responses infeasible. Instead, RFC4034 [5] provided a solution for precomputed denial-of-existence, by defining the NSEC record as follows: a lexicographic ordering of the names present in a zone is prepared, and every consecutive pair of names is signed; each pair of names is an NSEC record. Then, to prove the non-existence of a name (`x.example.com`), the nameserver returns the precomputed 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.[2] While this solution elegantly eliminates the need to trust the nameserver and allows for precomputation, it unfortunately allows for trivial *zone enumeration attacks*; namely, an adversary can use NSEC records to enumerate all the domain names present in the zone.

Why is zone enumeration a problem? This question has created some controversy, with many in the DNS community initially arguing that it is actually *not* a problem (*e.g.,* RFC 4033 [3]), before eventually arriving at consensus that it is a problem for some zones (RFC 5155 [30]). Zone enumeration allows an adversary to learn the IP addresses of all hosts in a zone (including routers and other devices); this information can then be used to launch more complex attacks, some of which are mentioned in RFC 5155:

> [T]he NSEC RR ... introduces a side-effect in that the contents of a zone can be enumerated. This property introduces undesired policy issues. ... An enumerated zone can be used, for example, 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. Many registries therefore prohibit the copying of their zone data; however, the use of NSEC RRs renders these policies unenforceable.

Indeed, some zones (*e.g.,* .de, .uk) require protection against zone enumeration in order to comply with European data protection laws [42], [1, pg. 37].

Thus, RFC 5155 [30] suggested NSEC3, a precomputed denial of existence technique, designed to make zone enumeration more difficult. With NSEC3, each domain name present in a zone is cryptographically hashed, and then all the hash values are lexicographically ordered. Every consecutive pair of hashes is an NSEC3 record, and is signed by the authority for the zone. 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.[3]

While hashing does make zone enumeration more difficult, NSEC3 is still vulnerable to zone enumeration via offline dictionary attacks. An adversary can query several random non-existent names, obtain a number of NSEC3 records, and then use rainbow tables or other offline dictionary attacks (for hash cracking) to determine the names that are present in the zone from the hashes in the NSEC3 records. Indeed, Bernstein's nsec3walker tool [13] does just that, effectively checking up to 2^{34} hash value guesses in one day, using a standard laptop and existing cryptographic libraries, and recent work [46] used a GPU to reverse 64% of the NSEC3 hashes in the .com zone in 4.5 days. The zone enumeration attacks are also acknowledged in RFC 5155 (Sec. 12.1.1).

NSEC3 does use a salt value (using the NSEC3PARAM record) to blunt the impact of offline dictionary attacks; however, in contrast to password-hashing applications that assign *unique* salt to each user's password, RFC 5155 states that "there MUST be at least one complete set of NSEC3 [records] for the zone using the *same* salt value." This is necessary to ensure that every possible query for a non-existent name properly maps to an NSEC3 record; if a different salt is used for each NSEC3 record, a query for a non-existent name might not map to *any* NSEC3 record. Moreover, since changing the salt requires re-computing the signatures for the entire zone, RFC 6781 [29] recommends updating the salt only when key-rollover takes place (an infrequent—monthly, or even yearly—event), which makes the salt a fairly weak defense against dictionary attacks. Moreover, once an adversary has collected a number of NSEC3 records and the salt for the zone, it can use offline dictionary attacks to learn the records present in the zone, even after the salt changed.

C. Our Model

Today, DNSSEC deployments support NSEC and/or NSEC3 and remain vulnerable to zone enumeration attacks. In this paper, we use cryptographic lower bounds to explain why zone enumeration attacks could not be addressed by previous designs, and propose a new solution, called NSEC5, that protects against them.

Our first contribution is the following cryptographic model:

Model. We have a trustworthy source, called a *primary nameserver*, which is trusted to determine the set R of names (`www.example.com`) present in the zone and their mapping to corresponding values ("155.41.24.251"). *Secondary nameservers* receive information from the primary nameserver, and respond to DNS queries for the zone, made by *resolvers*.

[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`) [40].

[2] For simplicity of exposition, we ignore the issues of wildcard records and enclosers in our descriptions of NSEC and NSEC3; see RFC 7129 [24].

[3] There was also an Internet Draft [23] (that expired without becoming an RFC) proposing NSEC4. NSEC4 combines NSEC and NSEC3, allowing zones to opt-out from hashed names to unhashed names. Like NSEC3, NSEC4 is vulnerable to zone enumeration via offline dictionary attacks.

Our denial-of-existence mechanism should achieve:

(1) Soundness. The primary nameserver is trusted to determine the set R of names in the zone, and to respond correctly to DNS queries. However, secondary nameservers and network adversaries are *not* trusted to respond correctly to DNS queries. Soundness ensures that bogus responses by secondaries or network adversaries will be detected by the resolver. This is the traditional DNSSEC security requirement of "data integrity and ... origin authentication" from RFC 3833 [7]; we require it to hold *even if secondary nameservers are compromised.*

(2) Privacy. Both primary and secondary nameservers are trusted to keep the contents of R private. (If they don't, there is nothing we can do, since they already know R.) However, resolvers are not. The privacy property must ensure that the response to a query by a resolver must only reveal information about the queried domain name, and no other names. Our main definitional contribution is formalizing the requirement of avoiding zone enumeration per RFC 5155 [30].

(3) Performance. We would like to limit the online computation that must be done by a nameserver in response to each query. This is discussed in *e.g.,* RFC 4470 [47].

The formal cryptographic model and security definitions are in Section II. We call a system satisfying these definitions a Primary-Secondary-Resolver (PSR) system.

D. Cryptographic Lower Bound

Section IV proves (in the random oracle model) that if the resolvers send queries in the clear (as they currently do in DNSSEC), then satisfying both soundness and privacy implies that nameservers must *necessarily* compute a public-key cryptographic signature for each negative response. This explains why NSEC and NSEC3, which limit the online computation at nameservers to cryptographic hashes, cannot prevent zone enumeration.

We also show that this problem cannot be solved on the resolver's end of the protocol: we show that even if the resolvers pre-process the query, then resolver-to-secondary-nameserver protocol is *necessarily* a secure public-key authentication (PKA) protocol [19, Sec. 3.5], for which the best known solution is a cryptographic signature anyway. Moreover, we show that a weakening of the soundness requirements—requiring soundness only against network attackers, but not against malicious or compromised secondary nameservers—still requires a secure PKA protocol. In Section IV-C we discuss whether our *privacy* requirements are "too strong", and argue that any meaningful relaxation, from a security standpoint, still implies PKA. Thus we conclude that preventing zone enumeration requires substantial ("public-key") online computation, rather than just private-key computation, *e.g.,* evaluating a cryptographic hash as in NSEC3.

E. NSEC5: A Denial-of-existence Mechanism

Armed with the knowledge that privacy necessitates one online public-key computation for every negative response, Section III-A presents NSEC5, a construction that matches our lower bound by requiring one online RSA computation for each negative response. NSEC5 provably achieves soundness (even in the presence of compromised secondary nameservers) and privacy against zone enumeration.

In designing NSEC5, our key observation is that we can "separate" our two security goals (soundness and privacy) using two separate cryptographic keys. To achieve soundness, we follow the traditional approach used in NSEC and NSEC3, and allow only the primary nameserver to know the primary secret key SK_P; this zone-signing key ensures the soundness of the zone. However, we now make the crucial observation that, while the soundness definition does not allow us to trust the secondary nameserver, our privacy definition does (because if the secondary nameserver is untrusted, then privacy is lost anyway, since it knows the entire zone). Thus, we achieve privacy by introducing a secondary key SK_S, that is known to *both* the primary and secondary namesevers. The secondary key is *only* used to prevent zone enumeration by resolvers, and will have no impact on the soundness of the zone. The public keys PK_P and PK_S corresponding to SK_P and SK_S will, naturally, be provided to the resolver, using the standard mechanisms used to transmit public keys in DNSSEC.

Construction. Our NSEC5 construction is extremely similar to NSEC3: we just replace the unkeyed hash used in NSEC3 with a new "keyed hash" F that uses the secondary keys PK_S, SK_S. Our solution is as follows.

The secondary keys $PK_S = (N_S, e_S)$ and $SK_S = (N_S, d_S)$ are an RSA key pair. For each record x present in the zone R, the primary nameserver computes a deterministic RSA signature on x using h_1 (where h_1 is a hash function, modeled as random oracle [10], that produces outputs one bit shorter than the bit-length of N_S, and can be implemented *e.g.,* by the industry-standard MGF [8, Sec. 10.2])

$$S(x) = (h_1(x))^{d_S} \bmod N_S \tag{1}$$

and hashes it to a short string with another hash function h_2 (*e.g.,* SHA-256, also modeled as random oracle)

$$F(x) = h_2(S(x))\,.$$

The resulting F values are lexicographically ordered, and each pair is signed by the *primary nameserver* using its key SK_P (as in NSEC and NSEC3). The resulting pair of F values is an NSEC5 record.

To prove the non-existence of a name q queried by the resolver, the secondary nameserver computes the "proof" value $\pi = S(q)$ using SK_S, and responds with (1) an NSEC5PROOF record containing proof π and (2) the signed NSEC5 record for the pair of hashes lexicographically before and after $h_2(\pi)$.

The resolver can then validate the response by (1) confirming that the NSEC5 record is validly signed by SK_P (using PK_P), (2) using PK_S to preform an RSA verification on the proof value π in the NSEC5PROOF, *i.e.,* checking that

$$\pi^{e_S} \bmod N_S = h_1(q)$$

and (3) checking that $h_2(\pi)$ (from the NSEC5PROOF) is lexicographically between the hashes in the NSEC5 record. Thus, the proof value $\pi = S(q)$ maintains soundness by proving that $F(q)$ is the correct "keyed hash" of q.

Note that the keyed hash $F(q)$ must be a deterministic and verifiable function of q. Our specific choice of the RSA signature algorithm used to compute S in equation (1) is thus crucial; in contrast, any secure signature algorithm can be used to sign the NSEC5 record using SK_P.

Privacy. In Section III-C we prove that our construction satisfies soundness and privacy as defined in Section II. Roughly, privacy follows because the resolver does not know the secondary secret key SK_S. This eliminates zone enumeration via offline dictionary attacks, since the resolver cannot compute the "keyed hash value" $F(q)$ on its own; the only way it can learn $F(q)$ is by asking online queries to the nameserver (or by breaking RSA!). The resolver's knowledge of the secondary public RSA key PK_S don't help either, since NSEC5 records do not contain RSA values, but rather *hashes* of RSA values.

Secret keys at nameservers? NSEC5 requires secondary nameservers to hold a *secret* secondary key SK_S. Fortunately, SK_S only needs to be as secure as the records whose the privacy it protects, since leaking SK_S does *not* compromise soundness in any way. Specifically, if SK_S is leaked or the secondary nameserver becomes adversarial, the soundness of the zone is not compromised; all that is lost is privacy against zone enumeration, effectively downgrading the security of NSEC5 to that of NSEC3. (See Section III-C.)

Soundness is maintained because only the primary nameserver can sign NSEC5 records; the resolver can use the secondary public key PK_S to verify that the secondary nameserver correctly computed $S(q)$ in the NSEC5PROOF, and responded with the right NSEC5 record. If an adversary wanted to send a bogus non-existence record, it would not be able to produce a properly-signed NSEC5 record covering $F(q)$, even if it knew the secret secondary key SK_S.

Deployment. Because NSEC5 is structurally very similar to NSEC3, it can incorporate performance or policy optimizations developed for DNSSEC, including opt-out and the space-saving techniques of NSEC4 [23]. Moreover, NSEC5 allows resolvers to verify using the same technologies they always used: hashing and validation of RSA signatures. We discuss other practical issues in Section III-B.

NSEC5 vs existing solutions. NSEC5 requires a single online RSA computation at the secondary nameserver, making it computationally heavier than NSEC and NSEC3. However, our lower bounds prove this extra computation is necessary to eliminate zone enumeration. In fact, online signing for denial of existence has already been standardized in RFC 4470 [47], further discussed in RFC 4471 [44], and implemented in nameserver software like powerDNS [39, Sec. 4] and Phreebird [28]. Existing online signing solutions require every nameserver (even the secondary) to be given the primary key for the zone, and use it to produce online signatures to responses of the form "q is a non-existent domain". However, some criticize these solutions for compromising soundness (when a secondary nameserver is hacked or leaks its key, the adversary can send false DNS responses). In contrast, NSEC5 has the same computational overhead as existing online-signing solutions but without the same risks to soundness; NSEC5 preserves soundness even when the secret secondary key SK_S is leaked, since online signing is used only to look up the correct NSEC5 record, but *cannot* be used to produce a false negative response.

Summary. Our lower bounds show that a solution that provides both privacy against zone enumeration and a weak notion of soundness (where secondary nameservers are trusted) *necessarily* requires one *online* public-key cryptographic operation per query for a non-existent domain name. Our NSEC5 construction precisely matches our lower bound, and also provides strong soundness (even if secondary nameservers are malicious or compromised). We therefore believe NSEC5 presents an attractive alternative to NSEC3 and existing online signing solutions (*e.g.,* RFC 4470 [47]) for those zones requiring both strong soundness and privacy guarantees. Of course, requiring online public-key operations at nameservers increases computational overhead and the risk that nameservers will be targeted by denial-of-service attacks (seeking to exhaust computational resources). Thus, individual zone operators will need to decide if our strong privacy and soundness guarantees are sufficiently important to justify the deployment of NSEC5.

F. Organization & Contributions

The organization of this paper follows the summary above. Section II presents our model and security definitions; our main definitional contribution is in our notion of privacy. Next, we present NSEC5 in Section III-A and prove that it satisfies soundness and privacy. Section III-B discusses practical considerations for NSEC5. Section IV presents our cryptographic lower bounds that explain why NSEC5 requires online signing at the secondary nameserver. Our results are supported by standard cryptographic definitions (signatures, RSA, random oracles) in the Appendix.

G. Other related work

There are several tools and primitives in the cryptographic literature that are related to our work. The first is *zero-knowledge sets* (ZKS) and their generalizations [31], [37]. They provide a primitive where a prover can commit to a database, and later open and prove the value in the database to a verifier in a zero-knowledge fashion. One can use ZKS in our setting, where the resolver is the ZKS verifier, the primary nameserver is the ZKS prover that creates the commitment to the set, the secondary namesever is the online ZKS prover that provides online proofs to the verifier. However, we can't use the existing ZKS solutions as is, because even the best-known constructions of ZKS [17] are too inefficient to be practical for DNSSEC[4]. On the other hand, the requirements in a ZKS are very stringent, in that one does not trust even the *primary nameserver* (*i.e.,* the commitment to the database). In the DNSSEC setting, where the primary nameserver is trusted, this property is not necessary. By working in this less stringent setting, we are able to obtain more efficient constructions.

Outsourced data structures that come with soundness guarantees are also relevant (see e.g. [14], [33], [34], [45]). These data structures return an answer along with a proof that the answer is sound; "soundness" means that the answer is consistent with some trusted external information. We add

[4] The verifier in [17] must verify $\log |U|$ mercurial commitments, where U is the universe of elements, and each verification involves a 'public-key computation'.

a privacy requirement to ensure no information leaks about questions that were not asked. Privacy in this setting was also considered by [22], but for queries that ask about order of existing elements; in particular, no privacy is provided by [22] in case the queried element does not exist.

II. Model and Security Definitions

We define the new primitive, Primary-Secondary-Resolver Membership Proof system (PSR), with the goal of secure denial of existence while preventing zone enumeration. A PSR is an interactive proof system consisting of three parties. The *primary nameserver* (or simply *primary*) sets up the zone by specifying a set $R \subseteq U$, where R represents the existing domain names in the zone and U represents the universe of all possible domain names. In addition to R, the primary specifies a value $v(x) \in V$ for every $x \in R$ (where $v(x)$ represents *e.g.,* the IP address corresponding to domain name x). It then publishes public parameters PK for the zone, which are distributed via the usual DNSSEC mechanisms. The *secondary nameservers* (or *secondaries*) get PK and extra information I_S necessary to produce response to queries made by *resolvers*. The resolvers get PK. After the setup phase is complete, the secondaries and resolvers act as provers and verifiers of statements of the form "$x \in R$ and $v(x) = y$" or "$x \notin R$".

Following DNSSEC, we consider only two-round protocols, where a query is sent from the resolver to the secondary and a response is returned. More interaction is possible, but we do not consider it here. DNSSEC has resolvers send queries in the clear (*i.e.,* send x and get $v(x)$), which is also what our NSEC5 construction does (Section III-A). However, for generality, when defining our model and proving our lower bound in Section IV, we allow resolvers to transform the query x before sending; in particular, our model allows resolvers to keep state between query issuance and answer verification (although our NSEC5 construction does not need this).

A. Algorithms for the Parties in PSR Systems

A PSR system consists of four algorithms.

The *Setup* algorithm is used by the primary nameserver to generate the public parameters PK, which it publishes to all parties in the protocol, and the information I_S, delivered to secondary nameservers. A resolver uses the *Query* algorithm to generate a query for elements in the universe; it then sends this query to a secondary, who replies to a query using the *Answer* algorithm. The resolver finally uses *Verify* to validate the response from the secondary.

Definition II.1. Let U be a universe of elements and V a set of possible values. A Primary-Secondary-Resolver system is specified by four probabilistic polynomial-time algorithms $(Setup, Query, Answer, Verify)$:

$Setup(R, v(\cdot), 1^k)$ On input k the security parameter, a set $R \subseteq U$, a value function[5] $v : R \to V$, this algorithm outputs two strings: public parameters PK and the information I_S given to the secondaries.

$Query(x, PK)$ On input $x \in U$ and the public parameters PK, this algorithm outputs a query q. It also leaves state information for the Verify algorithm.

$Answer(q, I_S, PK)$ The algorithm gets as input a query q for some element $x \in U$, the information I_S produced by Setup, and the public parameters. If $x \in R$ then the algorithm outputs a bit $b =$ *'yes'*, the value $v(x)$, and a proof π for $x \in R$ and $v(x)$. Else it outputs $b =$ *'no'*, an empty v, and a proof π for $x \notin R$.

$Verify(b, v, \pi)$ The algorithm, which is given state information from the Query algorithm, including x and PK, gets a bit b, a value v (empty if $b =$ *'no'*), and the proof π. If $b =$ *'yes'* then it checks that the proof π validates that $x \in R$ and the value is $v(x)$. If $b =$ *'no'* it checks to validate that $x \notin R$. If the proof is correct it returns 1 and otherwise 0.

For simplicity, our definition above considers only the case where the set R is static; R is chosen when the primary sets up the zone and does not change afterwards.[6]

We will require the above four algorithms to satisfy three properties: Completeness, Soundness, and Privacy.

B. Functionality and Soundness

The requirement that the system be functional is called, as is traditional in interactive proof systems, *completeness*. When the different parties are honest and follow the protocol, then the system should work properly; that is, resolvers will learn whether names are in the set R or not. Moreover, even if the queries are chosen adversarially by someone who has access to all the secret information, the answers should be verifiable, except with negligible probability.

Definition II.2. ***Completeness:*** *For all probabilistic polynomial time adversaries A, for all $R \subseteq U$ and $\forall x \in U$,*

$$\Pr\left[\begin{array}{l} (PK, I_S) \overset{R}{\leftarrow} Setup(R, v(\cdot), 1^k); \\ x \overset{R}{\leftarrow} A(PK, I_S); \\ q \overset{R}{\leftarrow} Query(x, PK); \\ (b, v, \pi) \overset{R}{\leftarrow} Answer(q, I_S, PK) : \\ Verify(b, v, \pi) = 1 \end{array}\right] \geq 1 - \mu(k)$$

for a negligible function $\mu(k)$.

Soundness is the traditional DNSSEC notion of security; we require that even a malicious secondary cannot convince an honest resolver of a false statement with more than a negligible probability. This must hold even when the malicious secondary gets to choose R and v, then gets (PK, I_S), and finally chooses element $x \in U$ it wishes to cheat on, and its deceitful proof π.

[5] This function *e.g.,* maps domain names to their corresponding IP addresses.

[6] There are methods for handling changes to R that borrow from the CRL world (*e.g.,* [34]) but we chose not to concentrate on this here. Our completeness, soundness, and privacy definitions and proofs would need to change to accommodate dynamic and possibly adversarial changes to R. Our NSEC5 construction can, however, use techniques similar to NSEC and NSEC3 to deal with dynamic changes to R.

Definition II.3. ***Soundness:*** *for all probabilistic polynomial time stateful adversaries A we have*

$$\Pr\left[\begin{array}{l} (R, v(\cdot)) \overset{R}{\leftarrow} A(1^k); \\ (PK, I_S) \overset{R}{\leftarrow} Setup(R, v(\cdot), 1^k); \\ x \overset{R}{\leftarrow} A(PK, I_S); \\ q \overset{R}{\leftarrow} Query(x, PK); \\ (b', v', \pi) \overset{R}{\leftarrow} A(PK, I_S) : \\ Verify(b', v', \pi) = 1 \wedge \\ ((x \in R \ \wedge \ (b' = \text{`}\textit{no}\text{'} \ \vee \ v' \neq v(x))) \ \vee \\ (x \notin R \ \wedge \ b' = \text{`}\textit{yes}\text{'})) \end{array}\right] \leq \mu(k)$$

for a negligible function $\mu(k)$.

Our definition is strong because it ensures (up to negligible probability) that an adversary cannot find $x \in U$ violating completeness or soundness even if it has I_S, the information given to the secondaries.

C. Privacy: Preventing Zone Enumeration

In our setting, privacy means preventing zone enumeration. We want to make sure that resolvers do not learn too much about the elements in the set R, apart from the responses to their queries. We formulate this requirement with a strong notion that we call **f-zero-knowledge** (f-zk for short), where $f(R)$ is some information about the set which we can tolerate leaking to the resolvers. For example, our NSEC5 construction has $f(R) = |R|$ (the number of names in the set R).

We formulate f-zk by requiring every PSR system to have its own *simulator* algorithm, who can fool a resolver into thinking that it is communicating with a real secondary in a PSR system. The simulator must do this without access to the set R; instead it is only given $f(R)$, and *limited oracle access* to R—the simulator may only ask the oracle if element x is in R if the resolver explicitly queries the simulator for x. Despite these limitations, the simulator must still be able to "forge" a satisfactory response to every query sent by the resolver, such that the resolver cannot distinguish between (1) an interaction with a secondary in a real PSR system (who knows R), and (2) an interaction with the simulator (who only knows $f(R)$ and those elements x queried by the resolver). It follows that the resolver learns nothing about R from its interaction with the secondaries, apart from $f(R)$ and whether the elements x that it queried are in R or not; this further implies privacy against zone enumeration. We note that the use of simulators to prove that a protocol is zero knowledge is standard in cryptography; see [25, Ch. 4] for a comprehensive treatment. Later, in Section II-D we show that our f-zk notion implies a more "intuitive" security definition.

PSR Simulator. More formally, we define a PSR Simulator. Let SIM be a probabilistic polynomial time algorithm with *limited oracle access* to R, meaning that SIM can only ask the R-oracle if $x \in R$ (and if so, what is $v(x)$) when the adversary explicitly queries the simulator for x. Upon initializing, SIM receives $f(R)$ and outputs fake public parameters PK^*, fake secret information SK_{SIM} and the leaked information $f(R)$. Next, SIM receives queries from the resolver and needs to output a (simulated) proof of either $x \notin R$ or of $x \in R$ plus $v(x)$; to do this, SIM is allowed to query the R-oracle for the element x. The simulator's output (public parameters and proofs) should be computationally indistinguishable from the output generated by a real PSR system.

We divide this process into two phases. In the first phase, we refer to the resolver as "the adversary" (since the resolver is the adversary that wishes to enumerate the zone). This first phase requires the adversary to take part in an interactive protocol with either the simulator or a PSR system; the adversary does not know if it is talking to the real PSR system or the simulator. The interactive protocol starts by giving the adversary the public parameters, either generated by the real PSR system:

$$(PK, I_S, f(R)) \overset{R}{\leftarrow} Setup(R, v(\cdot), 1^k)$$

or by the simulator that generates fake parameters:

$$(PK^*, SK_{SIM}, f(R)) \overset{R}{\leftarrow} SIM^R(f(R), 1^k)$$

Next, the adversary starts issuing queries q_i (adaptively), based on the public parameters and previous responses to queries it received. If the adversary is talking to the simulator, the simulator responds to the queries with the answers (b_i, v_i, π_i) using the fake public parameters PK^* and the fake secret information SK_{SIM}. If the adversary is talking to the real PSR system, it responds to the queries with the answers (b_i, v_i, π_i) using the real parameters and information (PK, I_S). The adversary can verify responses using the public parameters.

The second phase starts after the interactive protocol ends; here, "a distinguisher" is required to distinguish whether the adversary was interacting with the simulator, or with the real PSR protocol.

We say that the system is f-zk if there exists a simulator such that for every adversary, there is no distinguisher who knows R and can distinguish with more than a negligible advantage between the two views containing the public parameters, $f(R)$, queries and responses which were generated by either the system or the simulator.

Definition II.4. Let the leaked info $f()$ be some function from 2^U to some domain and let $(Setup, Query, Answer, Verify)$ be a PSR system. We say that it is f-zero knowledge (f-zk for short) if it satisfies the following property for a negligible function $\mu(k)$:

There exists a simulator SIM such that for every probabilistic polynomial time algorithms Adv and distinguisher D a set $R \subseteq U$ and $v : R \rightarrow V$ the distinguisher D cannot distinguish between the following two views:

$$view^{real} = \{PK, f(R), q_1, (b_1, v_1, \pi_1), q_2, (b_2, v_2, \pi_2), ...\}$$
$$view^{SIM} = \{PK^*, f(R), q_1, (b_1, v_1, \pi_1^*), q_2, (b_2, v_2, \pi_2^*), ...\}$$

with an advantage greater than $\mu(k)$, even for D that knows R and v (the two views are generated by the protocols described above).

Remark. Those familiar with cryptographic definitions will note that our definition requires simulation to be online: there is no rewinding, and the number and nature of the queries to the R-oracle are restricted to the queries made by the resolver.

This means that our simulator has little power (in the sense that the simulator only has the oracle access and $f(R)$ to work with) but it still manages to provide indistinguishable proofs with overwhelming probability. The less power the simulator has, the harder it is to construct a valid simulator that can fool an adversary, the more meaningful the f-zk property. Our simulator definition thus makes our f-zk requirement stronger.

Also note that the concept of a simulator receiving some $f(R)$ may look similar to the definition of auxiliary-input zero knowledge (see [25, Ch. 4]), but it is different. In the latter, both the adversary and the simulator receive the same auxiliary information and the adversary must still distinguish between the two views. In our case, the construction itself leaks the information $f(R)$, and we would like to show that it doesn't leak any additional information. The auxiliary-input property can be incorporated into our definition as well, in case we would like our resolvers to have some prior information about the set R; still, the resolvers would not be able to gain any additional information on R besides $f(R)$ and the prior information they received.

D. Zero-knowledge Implies Hardness of Zone Enumeration

Next, we argue formally that our f-zero-knowledge property indeed prevents zone enumeration; that is, that a resolver in a PSR system cannot learn anything about the elements in R (*i.e.,* the domain names that are present in the zone R) except for those elements for which it explicitly queried. In fact, we now prove that our definition of f-zk implies even a very weak version of privacy against zone enumeration; specifically, we show that a resolver whose goal is to learn whether one of two known elements (*i.e.,* domain names) is in the zone R, cannot succeed if he is not allowed to explicitly query for those two elements. We call this security property *selective membership security*, and define it using a game-based security definition where the resolver wins if it guesses a bit correctly.

Definition II.5. ***PSR security against selective membership****. A PSR protocol is said to be ε-secure against selective membership under an adaptive chosen message attack if every probabilistic polynomial time algorithm A playing against a challenger wins the following game with probability at most $\frac{1}{2}+\varepsilon$:*

1) *The adversary A starts by sending the challenger a set $S \subseteq U$, two target elements $x_0, x_1 \notin S$ and a value function v for the elements in $S \bigcup \{x_0, x_1\}$.*
2) *The challenger defines $R = S \bigcup \{x_0\}$ with probability $\frac{1}{2}$ and $R = S \bigcup \{x_1\}$ otherwise. Next the challenger runs algorithm $Setup(R, v(\cdot), 1^k)$, sends the output PK to the adversary A and keeps I_S secret to himself.*
3) *Algorithm A mounts an adaptive chosen message attack by sending queries to the elements $y_1, .., y_m$, where the queries are $q_i = Query(y_i, PK)$ and $y_i \notin \{x_0, x_1\}$. The challenger responds with proper answers to all the queries: $A_1, .., A_q$.*
4) *Finally A outputs one bit g, with $g = 0$ if A believes that $x_0 \in R$ and $g = 1$ if it believes $x_1 \in R$.*

We say that A won the game if the bit g is the correct guess, i.e., *if $x_g \in R$.*

We show that a PSR that is f-zk for $f(R) = |R|$ is also secure against selective membership attacks for a negligible ε.

Theorem II.6. *Suppose that we have an f-zk PSR system $(Setup, Query, Answer, Verify)$ for $f(R) = |R|$ and μ_f is the bound on the advantage of the distinguisher in f-zk. Then, it is also ε-secure against selective membership under an adaptive chosen message attack, where $\varepsilon = 2 \cdot \mu_f$*

Proof: We will show that the two possible views the adversary can witness in the security game, the one where $R = S \bigcup \{x_0\}$ and the other where $R = S \bigcup \{x_1\}$, are computationally indistinguishable.

For any choice of $(S, v : R \to V, x_0, x_1)$ we define four views. We will show that all four views are indistinguishable from one another and that two of them correspond to the two views of the adversary in the security game (either $x_0 \in R$ or $x_1 \in R$). Thus we can conclude that an adversary cannot find the additional element $x_g \in R$ with a non-negligible advantage; if it could, the adversary could also distinguish between the two views.

For $j \in \{0, 1\}$ denote the view of an adversary in the security game when $x_j \in R$ as $view_j^{real}(S, v(\cdot), x_0, x_1)$ and denote the view when we switch from a secondary to the simulator as $view_j^{sim}(S, v(\cdot), x_0, x_1)$.

First let us see that the views $view_j^{real}(S, v(\cdot), x_0, x_1)$ and $view_j^{sim}(S, v(\cdot), x_0, x_1)$ are indistinguishable for $j \in \{0, 1\}$. According to the f-zk assumption, for every choice of $(R, v(\cdot))$ the view of any adversary communicating with the simulator is indistinguishable from that of the same adversary communicating with the real system, when both are given $f(R) = |R|$. The adversary chooses S and knows that $|R| = |S| + 1$ and the simulator and real system also know the size of R by that same logic. So an adversary playing the security game cannot distinguish between cases where it is communicating with the simulator and ones where it communicates with the real system with advantage greater than μ_f, according to the definition of the f-zk property, which makes those views indistinguishable.

Now we notice that the views $view_0^{sim}(S, v(\cdot), x_0, x_1)$ and $view_1^{sim}(S, v(\cdot), x_0, x_1)$ are not only indistinguishable, but identical. This is true because the simulator SIM doesn't know the full set R—SIM only knows $|R|$, and has limited access to an R-oracle that SIM may query for an element x only when the adversary explicitly queries SIM on x, but not for any other elements. Since the adversary may not query SIM for x_0, x_1 (because it is his target challenge), the adversary can send identical queries to SIM and get identical answers in both views. Moreover, both views are identically distributed during the key generation, since SIM gets the same $f(R)$ and cannot query its R-oracle. Thus, both views are identically distributed and cannot be distinguished.

Combining it all, we get that $view_0^{real}(S, v(\cdot), x_0, x_1)$ and $view_1^{real}(S, v(\cdot), x_0, x_1)$ cannot be distinguished with probability greater than $2\mu_f$. This means that any probabilistic polynomial time adversary can win the selective security game with only a negligible advantage of $2 \cdot \mu_f$. ■

$R = \{x_1, \dots, x_r\}$	Set of domain names in the zone
U	Universe of domain names
V	Universe of IP addresses
$v : R \to V$	Function mapping domain names in the zone to IP addresses
PK_P, SK_P	Primary (zone-signing) keys
$Sig(\cdot)$	Signature using primary key
$PK_S = (N_S, e_S)$	Secondary RSA public key
$SK_S = (N_S, d_S)$	Secondary RSA secret key
$h_1 : U \to \{0,1\}^{\lvert N_S \rvert - 1}$	Hash function (*e.g.,* MGF)
$h_2 : \mathbb{Z}_{N_S} \to \{0,1\}^n$	Hash function (*e.g.,* SHA-256)
$S : U \to \mathbb{Z}_{N_S}$	Function $(h_1(\cdot))^{d_S} \bmod N_S$
$F : U \to \{0,1\}^n$	Function $h_2(S(\cdot))$

Fig. 1. Table of notation.

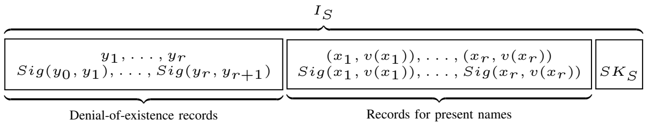
I_S
$y_1, \dots, y_r$
$Sig(y_0, y_1), \dots, Sig(y_r, y_{r+1})$
$(x_1, v(x_1)), \dots, (x_r, v(x_r))$
$Sig(x_1, v(x_1)), \dots, Sig(x_r, v(x_r))$
SK_S
Denial-of-existence records
Records for present names

Fig. 2. I_S, the extra information given to the secondaries.

III. NSEC5 Construction and Security Proof

Here we show why our NSEC5 construction is a secure PSR system, and prove its security. A high-level discussion of our NSEC5 construction was already provided in Section I-E. Section III-A presents NSEC5 in the context of our formal model of a PSR (per Section II), and Section III-C proves the security of NSEC5 in the random oracle model (per [16]). Table 1 summarizes our notation. Section III-B discusses practical considerations, including computational requirements, and the DNSSEC record types required by NSEC5.

A. NSEC5 as a PSR System

We specify our NSEC5 construction within the model of a PSR system in Section II.

Building blocks. Our NSEC5 construction is based on an RSA permutation, two hash functions, and a signature scheme. The RSA permutation has a key generation function that generates an RSA key pair $PK_S = (N_S, e_S)$ and $SK_S = (N_S, d_S)$. We also use two cryptographic hash functions: h_1 produces outputs one bit shorter than the bit-length of N_S (and can, *e.g.,* be the industry-standard MGF [8, Sec. 10.2]), and h_2 produces outputs of length n (n needs to be only as big as the security level we wish to achieve; see discussion of key length in Section III-C). Our security proofs model both h_1 and h_2 as random oracles per [16]. Finally, we use any signature scheme that provides existential unforgeability against chosen message attacks [27] (formally defined in Appendix B).

PSR algorithms. We now map our NSEC5 construction to the four PSR algorithms described in Section II.

Setup: The primary nameserver runs the setup algorithm $Setup(R, v(\cdot), 1^k)$, taking in the set R (*e.g.,* domain names in the zone) and its associated values v (*e.g.,* the IP addresses corresponding to those domain names), and security parameter k. It generates the public parameters PK and the information for the secondary nameservers I_S as follows.

It chooses the hash functions h_1 and h_2 and generates a key pair (PK_P, SK_P) for the existentially-unforgeable signature (the "primary keys") and an RSA key pair (PK_S, SK_S) (the "secondary keys"). The public parameters are $PK = (PK_P, PK_S, h_1, h_2)$.

Next, it constructs I_S, the information given to the secondary nameservers as follows: First, it signs the names that are present in the zone: using the primary secret key SK_P and the existentially-unforgeable signature algorithm, it obtains $Sig(x, v(x))$ for each element $x \in R$ (domain name) and its corresponding values $v(x)$ (IP address). Next, it constructs the authenticated denial-of-existence records: it uses the secondary secret RSA key $SK_S = (d_S, N_S)$ and the "full-domain" hash function h_1 to compute a deterministic RSA signature on each $x \in R$ as

$$\pi = S(x) = (h_1(x))^{d_S} \bmod N_S \qquad (2)$$

which is then hashed to a shorter string using h_2

$$y = F(x) = h_2(\pi) \qquad (3)$$

The y values are lexicographically ordered as $y_1, \dots, y_r$, and $y_0 = 0^n$ and $y_{r+1} = 1^n$ are added. For $j \in \{0, \dots, r\}$, each pair (y_j, y_{j+1}) is signed using the existentially-unforgeable signature algorithm with the primary secret key SK_P to obtain $Sig(y_j, y_{j+1})$.

Finally, the secondary nameserver is given the following information as I_S: the secondary secret key SK_S, the pairs $(x, v(x))$ and their signatures $Sig(x, v(x))$ for every $x \in R$ present in the zone, and the denial-of-existence pairs (y_j, y_{j+1}) and their signatures $Sig(y_j, y_{j+1})$ for $j = 0 \dots r$, see Figure 2.

Query: Resolvers send queries in the clear: $Query(x, PK)$ outputs element x (a domain name) as the query q.

Answer: Secondary nameservers run $Answer(q, I_S, PK)$ to respond to queries by resolvers. First, the secondary checks if $q \in R$. If so, it returns the corresponding signed records

$$\text{‘}yes\text{’}, (q, v(q)), Sig(q, v(q))$$

Otherwise, it uses the secondary secret key SK_S to compute the RSA signature $\pi_y = S(q)$ per equation (2), hashes this down to $y = h_2(\pi_y)$ per equation (3), finds the appropriate denial-of-existence record[7] by locating index j for which $y_j < y < y_{j+1}$, and returns

$$\text{‘}no\text{’}, (y_j, y_{j+1}), (\pi_y, Sig(y_j, y_{j+1})) \quad \text{where } \pi_y = S(q)$$

Verify: The resolvers verify the response with $Verify(b, v, \pi)$.

If the response had $b =$ ‘*yes*’, they use the primary public key PK_P to verify that $Sig(q, v(q))$ is a valid signature on $(q, v(q))$. If so, return ‘1’ for success; else return ‘0’.

Otherwise, $b =$ ‘*no*’. Resolvers then: (a) use the primary public key PK_P to verify that $Sig(y_j, y_{j+1})$ is a valid signature on (y_j, y_{j+1}) (b) use h_2 and π_y to check that

$$y_j < h_2(\pi_y) < y_{j+1}$$

[7] There is a negligible chance that such a record will not exist due to a hash collision; if that is the case, return "failure"; the same approach is used when a hash collision occurs in NSEC3 [30, Section 7.2.9]. We bound the probability of such an occurrence in Section III-C, where we evaluate the completeness of NSEC5.

and (c) use the secondary key $PK_S = (e_S, N_S)$ to verify that π_y is a deterministic RSA signature on q, *i.e.,* that

$$h_1(q) = \pi_y^{e_S} \bmod N_S \tag{4}$$

If all three checks pass, return '1'; else return '0'.

B. Practical Considerations and Compatibility with DNSSEC

We highlight the similarities and differences between our construction and the existing DNSSEC standard with NSEC3. To do this, we map the description in Sections I-E,III-A to DNSSEC record types, and discuss computational overheads.

Before we move on to less straightforward details of our construction, we note that the primary public key PK_P from Section III-A is the usual DNSSEC zone-signing key (ZSK), stored in a DNSKEY record and securely distributed using the usual DNSSEC mechanisms. Each pair $(x, v(x))$ of domain name $x \in R$ and IP addresses $v(x)$ present in the zone is the usual DNS A records; their signatures $Sig(x, v(x))$ are the usual DNSSEC RRSIG records. Each of our new NSEC5 records contains a pair (y_j, y_{j+1}) of lexicographically-adjacent hash values (see equations (2) and (3) above); each NSEC5 record is signed using the primary key (the ZSK) and its signature is stored in a DNSSEC RRSIG record. Observe that our NSEC5 records are almost identical to NSEC3 records.

Computational overhead. The main computational overhead of our approach over NSEC3 is online signing at the secondary nameservers. Specifically, we require secondaries to compute a deterministic RSA signature $S(q)$ *online* for every query q that requires a negative response (see equation (2)). Note, however, that online signing has been standardized (RFC 4470 [47]) and implemented in commercial DNSSEC systems [39, Sec. 4], [28]. Moreover, in Section IV we will prove that online signing is necessary to satisfy our security definitions.

We also require resolvers to verify the RSA signature $S(q)$ (equation (4)). This is no slower than actually verifying the signature on an NSEC record itself, representing no more than a 2x increase in computational overhead. Moreover, comparing this extra overhead to its analogous computation in NSEC3—namely, computing multiple iterations of a hash (*e.g.,* SHA-256) on the query q—suggests that NSEC3 and NSEC5 can have similar computational overhead at the resolver; specifically, RFC 5155 [30, Sec 10.3] specifies that the iterative NSEC3 hash computation can have up to the same cost as an RSA signature verification (*e.g.,* 500 SHA1 iterations for a 2048-bit RSA signature).

Finally, the primary nameserver also needs to compute some extra signatures when setting up the zone—an additional $r = |R|$ RSA signing computations (to compute the y_j's) in addition to the $2r + 1$ signatures needed to sign the NSEC5 records. This represents a 2x increase over NSEC3, which requires $|R| + 1$ signatures on the NSEC3 records.

Secrets at the nameservers. Secondary nameservers must hold the *secret* secondary key SK_S, which can be transmitted from the primary to the secondaries over some external secure channel. Fortunately, SK_S only needs to be as secure as the records whose privacy it protects, since leaking SK_S does not compromise soundness in any way (Section III-C).

New DNSSEC record types. Our NSEC5 solution requires the secondary to respond to queries for non-existent names with NSEC5 records (pairs (y_j, y_{j+})), as well as the RSA value $S(q)$. In contrast to the information in the NSEC5 record, which is computed during setup and signed by SK_P, the RSA value $S(q)$ is computed online and is not signed. We therefore propose transmitting $S(q)$ from secondary to resolver in a new *unsigned* record type (NSEC5PROOF).

We must also consider how the primary nameserver can authentically transmit the secondary public key PK_S and hash functions h_1, h_2 to the resolver and secondary nameserver. An analogous issue arises in NSEC3, when transmitting the salt and hash function to the resolver; NSEC3 deals with this by including the salt and an "algorithm identifier" for the hash in the NSEC3 record itself, and having the entirety of the NSEC3 record signed using the primary secret key SK_P [30]. We could analogously include the secondary public key PK_S and algorithm identifiers for the hash functions h_1 and h_2 in each NSEC5 record, and then sign the entire NSEC5 record using SK_P.[8] However, we may want to avoid including PK_S in each NSEC5 record (since a 2048-bit public RSA key is much larger than the 24-bit NSEC3 salt). NSEC3 uses the NSEC3PARAM record to transmit the NSEC3 salt and hash function to the secondary nameservers; we could similarly distribute PK_S, h_1, h_2 in an NSEC5PARAM records. Alternatively, we might use a signed DNSKEY record with flag bits set to indicate that PK_S is *not* a signing key for the zone[9]. The DNSKEY or NSEC5PARAM would be signed by the primary key and their signatures stored in RRSIG records.

Opt-out. Finally, the structural similarity of NSEC5 with NSEC3 and NSEC allows for easy adoption of existing DNSSEC mechanisms such as wildcards, Opt-out [23], [30] and Opt-in [6]. We will address these details in future work.[10]

C. Proof of Security for NSEC5

We show that our system is complete, sound, and leaks nothing more than the size of the set R. Our proof is in the random oracle model (using "programmable" random oracles [16], [36]). Note that we assume that the set R is chosen independently of the random oracles—that is, hash values did not influence the set names in the zone R (this assumption can be removed at the cost of slightly complicating the proof and making the reduction less tight).

Theorem III.1. *The four algorithms described above constitute an f-zk PSR for the function $f(R) = |R|$.*

Proof: We need to show the three properties in Definitions II.2, II.3 and II.4.

Completeness. The only way completeness can be violated is if there is a collision $F(x) = y_j$ for some $x \notin R$ (see

[8] Notice that by signing the entire NSEC5 record with SK_P, which is only known to the primary, we ensure that PK_S, h_1, h_2 are authentically communicated from the primary to the resolver.

[9] This approach complies with RFC 4035 [4]'s discussion on the use of the DNSKEY record for non-zone signing keys.

[10] Using the wildcard optimization devised for NSEC4 [23], our denial-of-existence response will contain two NSEC5 records and two NSEC5PROOF records in the worst case. This makes it only about one RSA-value (2048 bits) longer than today's unoptimized NSEC3 standard with 2048-bit RSA, which contains three NSEC3 records in the worst case.

Footnote 7). For every $R \subseteq U$ we will show that after we run $Setup(R, v(\cdot), 1^k)$ to get (PK, I_S) and let the adversary A pick and query x, the $Verify$ algorithm will output 0 with probability at most $(q_h+2)(r+2)2^{-n}$, where q_h is the number of hash (random oracle) queries made by A.

Let us assume that A makes an h_1-query on x and an h_2-query on $S(x)$ before asking for a query on x (any A can be easily modified to do so, at the cost of at most two additional random oracle queries, because A knows SK_S). Denote by q_{h_1} and q_{h_2} the numbers of queries A makes to its h_1 and h_2 oracles, respectively, after this modification. Consider every query A makes to h_1 for any $x \notin R$. The probability that the output of this query is equal to $h_1(x_j)$ for some $x_j \in R$ is at most $r2^{-|N_S|+1}$ (recall that $r = |R|$). The probability that there exists an h_1 query that caused this event to happen is at most $q_{h_1} r 2^{-|N_S|+1}$. Suppose such an event did not happen. Now consider every query A makes to h_2 on input $S(x)$ for any $x \notin R$. Since RSA is a permutation and h_1 collisions did not occur, $S(x) \neq S(x_j)$ for any $x_j \in R$. So the probability $h_2(S(x))$ collides with any y_j is at most $(r+2)2^{-n}$. Hence, the probability of a collision is at most $q_{h_1} r 2^{-|N_S|+1} + q_{h_2}(r+2)2^{-n} \leq (q_h+2)(r+2)2^{-n}$.

Remark. We can also consider a weaker adversary who is not given SK_S but can make q_S queries to a secondary instead. Such an adversary cannot obtain $S(x)$ on his own without forging an RSA signature and is thus forced to perform an on-line, instead of an off-line, attack. We can therefore show (using the full-domain-hash reduction of [18]) that, under the (t, ε) RSA assumption (Appendix C), the probability such an adversary violates completeness in time t is approximately $q_S(\varepsilon + r2^{-n})$, where q_S is the number of active queries to the secondary. Passive random oracle queries are of no value to such an adversary.

Soundness. The soundness of our proposal follows via a tight reduction from the existential unforgeability of the underlying signature scheme that is used to sign records with the primary key SK_P. More formally, given an adversary A that breaks soundness with probability ε (per Definition II.3), we can construct a forging adversary B that, given a public key PK_P, can forge digital signatures (or, more formally, win the existential unforgeability game defined in Appendix B) with probability ε, where both algorithms have similar running times. In the existential unforgeability game for digital signatures, the forger B can ask a signing oracle for signatures on arbitrary messages and must forge a signature on a new message.

We construct the forger B as follows. Initially the forger B receives the public key PK_P (and the security parameter k) from the challenger. It then starts running adversary A that breaks soundness of our NSEC5 scheme. First, the forger B obtains the set R and the function $v(\cdot)$ that is output by A (See Definition II.3). Next, it needs to compute I_S that it will return to A. To do this, the forger B runs the NSEC5 $Setup$ algorithm in Section III-A, with the following modifications:

- The forger B no longer generates the primary keys; instead, PK_P (that B was given as input) is used as the primary public key.
- The forger B does not have the private primary key SK_P. Instead, it queries its signing oracle to obtain signatures on pairs (y_j, y_{j+1}) for $j \in \{0, \dots, |R|+1\}$ and pairs $(x, v(x))$ for all $x \in R$.

The forger then gives A the output of the $Setup$ algorithm. A then outputs the value x that it wishes to cheat on, and corresponding values b', v', π. Suppose $Verify(b', v', \pi) = 1$.

Suppose that $b' =$ *'yes'* but $x \notin R$ or $v(x) \neq v'$ so that adversary A wins the soundness game for the NSEC5 scheme. B can output the message (x, v') and its signature as the forgery; B wins the game because this signature was not requested by B during setup.

Suppose now that A wins the soundness game because $b' =$ *'no'* but $x \in R$. Since π passes verification, it contains π_y that satisfies equation (4). Also, π contains a signature σ computed using the primary key on some pair (y_1', y_2'), such that $y_1' < h_2(\pi_y) < y_2'$. We need to show that the forger B did not request the signature σ when it ran its' modified NSEC5 $Setup$ algorithm; thus the forger B can win the game by outputting message (y_1', y_2') and σ as its forged signature. First, observe that $x \in R$, so it follows that when B ran its modified NSEC5 $Setup$ algorithm, it computed the RSA value $S(x)$ per equation (2). Next, because the RSA function $S(\cdot)$ (see equation (2)) is deterministic, it follows that $\pi_y = S(x)$. Therefore, value $y = h_2(\pi_y)$ must be one of the values in the sequence $y_0, \dots, y_{|R|+1}$ that were used to construct denial-of-existence records during setup. Since $y_1' < y$ and $y_2' > y$, it follows that (y_1', y_2') are not adjacent values. We conclude that the forger B never queried its signing oracle for a signature on (y_1', y_2') during setup.

Therefore, we see that B succeeds whenever A succeeds in breaking soundness, *i.e.,* with probability ε.

Privacy. In order to show that NSEC5 is f-zk for $f(R) = |R|$, we need to construct a suitable simulator, for which no probabilistic polynomial time distinguisher can distinguish between an interaction with the simulator and one with the real NSEC5 system. The indistinguishability of our simulator will follow via a tight reduction from the (t, ε)-RSA assumption (Appendix C).

The simulation relies on the verifiable pseudorandomness of the function F in equation (3), which is captured in the following lemma.

Lemma III.2. *For every $x \in U$, the value $F(x)$ is pseudorandom over $\{0,1\}^n$ in the following sense: no probabilistic polynomial-time adversary, who gets the public key (N_S, e_s) and the value x, and can ask for $F(x_i)$ and $S(x_i)$ on any sequence of points $x_1, x_2 \dots$ not containing x, can distinguish $F(x)$ from a random value in $\{0,1\}^n$ with non-negligible advantage. More precisely, an adversary who has advantage ε in distinguishing $F(x)$ from random in time t can be used to break the (t', ε) RSA assumption (stated in Appendix C) with $t' \approx t$.*

Note that this means that the function $F()$ combined with $S()$ constitutes a selective VRF, as defined in [32]. This is a very simple and efficient implementation of the primitive (albeit, only in the random oracle model); see also [35].

Proof: Assume to the contrary that there exists such an adversary A that has non-negligible advantage ε. Because h_2

is random and $F(x) = h_2(S(x))$, A has zero advantage unless it queries h_2 on $S(x)$, either directly or through one of its F queries. Hence, the probability that $h_2(S(x))$ is queried during the attack by A is at least ε. It remains to show that $S(x)$ is hard for A to compute, which we do similarly to [10].

Specifically, we show that, using A, we can invert the RSA permutation with probability ε, violating the RSA hardness assumption as stated in Appendix C. Recall that we get to determine ("program") the random oracle outputs, as long as they are uniform.

We are given a public RSA key (N, e) and challenge z that we wish to invert. If the bit-length of z is strictly less than the bit-length of N, set $z' = z$. Else, repeatedly try a random $\alpha \in \mathbb{Z}_N^*$ until $z' = z\alpha^e \bmod N$ has bit-length less than the bit-length of N (the expected number of tries is less than 2). Observe that if we invert RSA on z', we will also invert it on z by dividing the answer by α.

Assume wlog that A makes at most Q queries, that it does not repeat queries, and that for every $x_i \neq x$ for which A asks the random oracle to evaluate $h_1(x_i)$, it also asks to see $(F(x_i), S(x_i))$. Before running A, we do the following for each i between 1 and Q: draw a uniformly random $c \in \mathbb{Z}_N$ and check if the bit-length of $c^e \bmod N$ is strictly less than the bit-length of N. If not, try a fresh random c (the expected number of attempts is less than two). If yes, let $c_i = c$ and $z_i = c^e \bmod N$. Now we run A on the public key (N, e) and any x. Every time A queries the h_1 random oracle on $x_i \neq x$, we answer with z_i (*i.e.,* program $h_1(x_i) = z_i$); we also return $S(x_i) = c_i$, draw a uniformly-random value r_i, return $F(x_i) = r_i$, and program $h_2(c_i) = r_i$.

When h_1 is queried on x, we answer with z'. This ensures that $z' = S(x)^e \bmod n$. Queries to h_2 are answered consistently if h_2 has already been programmed on that query, and randomly otherwise. The distribution A witnesses is identical to the real distribution. Hence, the probability that the h_2 table contains $S(x)$, which is the RSA inverse of z', is ε. ■

Corollary III.3. *For every set $R \subset U$, an adversary who gets (N, e) as input, runs in time t and has advantage ε in distinguishing $\{F(x)|x \in R\}$ from a set of $|R|$ random values in $\{0,1\}^n$ can be used to break the (t', ε')-RSA assumption, where $\varepsilon' = \varepsilon - |R|/N$ and $t' \approx t$. This holds even if the adversary can query for any $x_i \notin R$ to obtain $F(x_i)$ and $S(x_i)$.*

This corollary follows because of random self-reducibility of RSA: inverting RSA on any value in a set of r random values is essentially as hard as inverting it on a single element (see Appendix C). Here we use specific properties of RSA; if we had used a generic trapdoor permutation in our construction, we would have lost a factor r in the advantage due to a hybrid argument (the only other place we use specific properties of RSA is in the proof of Lemma III.2, where we save a factor of 2 by rerandomizing z so it has the correct bit-length).

We now show the simulator. Recall from Section II-C that the simulator is given oracle access to R (*i.e.,* the set of domain names in the zone). Our simulator algorithm $SIM^R(1^k, 1^{|R|})$ is as follows. Like the primary nameserver, SIM chooses the hash functions h_1 and h_2, generates the primary keys (PK_P, SK_P) and the secondary keys (PK_S, SK_S), and outputs $PK^* = (PK_P, PK_S, h_1, h_2)$. SIM then selects a list of $|R|$ uniformly random values in $\{0,1\}^n$, sorts them lexicographically to obtain $y_1, \ldots, y_{|R|}$, and adds the values $y_0 = 0^n$ and $y_{r+1} = 1^n$. For $j \in \{0, \ldots, |R|\}$, each pair (y_j, y_{j+1}) is signed using SK_P to obtain $Sig(y_j, y_{j+1})$.

Next, SIM receives queries from the resolver and outputs a (simulated) proof for either $x \notin R$ or $x \in R$ plus $v(x)$ as follows. For each received query x_i, SIM uses his oracle access to the set R to check if either $x_i \notin R$ or $x_i \in R$ and if so to obtain $v(x_i)$. If $x_i \in R$, SIM uses the secret key SK_P and the existentially-unforgeable signature scheme to produce a signature $Sig(x_i, v(x_i))$, and outputs

$$\text{‘}\textit{yes}\text{’}, (x_i, v(x_i)), Sig(x_i, v(x_i))\,.$$

SIM remembers the answer and outputs the same answer in case of repeated queries. Otherwise $x_i \notin R$, and SIM does the same as the NSEC5 secondary nameserver: it uses the secondary secret key SK_S to compute the RSA signature $\pi = S(x_i)$ according to equation (2) and the hash value $y = h_2(\pi)$ per equation (3); it then searches for an index j for which $y_j < y < y_{j+1}$. If such an index j is found, SIM returns

$$\text{‘}\textit{no}\text{’}, (y_j, y_{j+1}), (\pi, Sig(y_j, y_{j+1}))$$

If no such index j is found, *i.e.,* a collision has occurred, SIM returns "failure," just like the real NSEC5.

Now we need to show that the view of the adversary communicating with SIM is indistinguishable from that of the adversary communicating with the real NSEC5 system. The only difference between the outputs of SIM and the outputs of real NSEC5 system is in the values y_j, which are random for the simulator rather than computed by applying equations (2),(3) to the elements of R. Fortunately, the adversary cannot query for $F(x)$ and $S(x)$ for $x \in R$ (because those queries give positive, rather than negative, proofs, and positive proofs do not use $F()$ and $S()$). Thus Corollary III.3 can be used to conclude the proof. ■

Key Length. We discuss key lengths requirements for the security of NSEC5. First, observe that RSA modulus N_S for the secondary key must be long enough to withstand attacks for as long as the privacy of the zone is important. Even if the secondary key is rolled frequently, it cannot be too short, because zone privacy may be important for a period longer than the lifetime of the key. Meanwhile, the primary key (PK_P, SK_P) (*i.e.,* the zone-signing key) is used for authentication, but not privacy; thus, it needs to remain secure only during its lifetime.

Next, consider n, the length of the output of the hash function h_2, which impacts the length of the NSEC5 response and storage (of I_S) at the secondary nameservers. Length n does not affect soundness or privacy, but it does affect completeness, *i.e.,* the probability that the secondary returns "failure" rather than a proof for $x \notin R$. Since secondaries can fail for a variety of non-NSEC5-related reasons, n needs to be only large enough to make completeness violations much less likely than other server failures. The choice of n thus depends on the security model for completeness. If everyone is honest and there are at most q_S online queries to the nameserver, completeness will be violated with probability about $|R|q_S2^{-n}$, so even $n = 80$ could suffice. As we explained in the remark in the completeness section of the proof of Theorem III.1, the same holds even if the queries are adversarial (for example, are given to a resolver by an adversary), but the adversary does not know the secret secondary key SK_S. On the other hand, an adversary who knows SK_S can find a query that violates completeness with probability about $|R|q_H 2^{-n}$, where q_H is the number of offline hash computations the adversary can make, which can be considerably greater than q_S. If completeness violation by such an adversary is a concern, then $n = 128$ seems more prudent. Perhaps of a greater concern is an adversary who can influence the set R to make completeness violations more likely (our formal models do not consider such an adversary, because they suppose the set R is static, rather than dynamically modified by the adversary). If such an adversary has no access to SK_S, then it would violate completeness with probability about $q_S^2 2^{-n}$ (by an argument similar to the one in the remark in the proof of completeness above), and $n = 128$ seems sufficient. If the adversary can influence R (*i.e.,* can control the names that go in the zone) and knows SK_S, then it will be able to violate completeness with probability at most $q_H^2 2^{-n}$ by a standard birthday bound argument, and $n = 200$ is more appropriate. Thus, a conservative choice is n large enough to prevent birthday attacks (*e.g.,* letting h_2 be SHA-256 with $n = 256$ for a security level of 128 bits). However, this may be overkill; since the choice of n only affects completeness it may be worthwhile to design for a weaker threat models that results in a smaller choice of n.

Perfect completeness. We could get perfect completeness for NSEC5 by adding proofs for the special case of collisions in the hash functions h_1 and h_2. If $x \notin R$ collides with $x^* \in R$ ($F(x) = F(x^*)$) then a valid proof for $x \notin R$ would be $(\text{'}no\text{'}, (S(x), S(x^*), x^*, v(x^*), Sig(x^*, v(x^*))))$. A resolver would verify this proof by checking that $F(x) = F(x^*)$ and verifying the signature $Sig(x^*, v(x^*))$. Changing the scheme in this way would leak information about R (*i.e.,* that $x^* \in R$ and $v(x^*)$), but would not violate privacy since collisions occur with negligible probability for large n.

IV. Online Public-Key Operations are Necessary

We now show that secondary nameservers in a PSR system must perform a public-key computation *on every query* for a non-existent name. We do so by showing that *any* PSR system can be used to construct a public-key signature scheme, where the complexity of the signer in the signature scheme is roughly equal to the complexity of the secondary nameserver (plus the complexity of the query algorithm) in the PSR. The signature scheme will be secure as long as the PSR system is complete, sound, and private—even if it satisfies only (1) the weaker privacy notion of Definition II.5 rather than full-fledged zero-knowledge of Definition II.4, and (2) a weaker notion of soundness that robust to network attackers but *not* to compromised secondary nameservers. (This weak soundness adversary is an attacker that can ask secondary nameservers to respond to queries, but does not have the information I_S stored at secondary nameservers.)

In a public-key signature system, signers must perform a public-key operation for each message they sign; thus, by constructing a signature scheme from a PSR system, we have shown that the same holds for a PSR. Of course, a limited number of signatures can be precomputed in any signature scheme, and the same holds for a PSR (*e.g.,* all positive responses may be precomputed, as in most existing constructions, including NSEC5). The rest—and, in particular, negative responses to unexpected queries—must be done online.

We obtain signatures only when the PSR system satisfies the following property: the query algorithm is either (1) deterministic or, (2) randomized such that soundness holds even when the adversary has the knowledge of the random values used to generate the query q. Note that this property is trivially satisfied by systems like NSEC5 and the current DNS/DNSSEC standards where $q = x$ (*i.e.,* the query is identical to the queried name).

However, we also show that a more complicated query algorithm will not help much. Even when the PSR system does not satisfy this property, we obtain an interactive protocol that is very similar to signatures: namely a public-key authentication protocol (PKA), in which the sender transmits an authenticated message to the receiver using some interaction. Again, in our obtained PKA protocol, the complexity of the sender is similar to that of the secondary nameserver in a PSR system. Since such a PKA protocol is not known to have any implementation that is much more efficient than a digital signature scheme, we can conclude that a non-trivial computational task is required of secondaries in PSR systems.

This explains why approaches like NSEC3—that provide strong soundness (robust to compromised secondary nameservers), but avoid online "public-key" computations—cannot prevent zone enumeration. It also explains why zones requiring strong privacy still have to use online signing (*e.g.,* RFC 4470 [47]) even if they require only weak soundness (robust to network attackers but not compromised nameservers).

Section IV-A presents results on transforming PSR schemes to PKA schemes, and Section IV-B extends this to signatures when the constraints on the query algorithm are satisfied. Both our transformations are in the random oracle model. Section IV-C argues that any meaningful relaxation to our privacy requirement still implies PKA.

A. Public-Key Authentication from PSR

Defining Public-key Authentication Security Public-key authentication (PKA; see [19, Section 3.5]) can be seen as a relaxation of signature schemes in which we tolerate interaction between the sender and the receiver, and we give up the transferability property (*i.e.,* the receiver is no longer able to convince a third party that the signature is valid).

PKA schemes are related to, but are harder to build than *identification protocols*, in which there isn't even a message; instead the prover (sender) convinces the verifier that he is present. (Such protocols can be used, *e.g.,* for access control where key cards act as provers.) Identification protocols can be constructed from any zero-knowledge proof of knowledge [20] for a computationally-hard problem, but in practice, there is no known PKA or identification protocol where the efficiency of the prover is better than a signer in a signature scheme.

We now define selective and existential security for PKA.

Definition IV.1. ***Selectively-secure PKA.*** *A public-key authentication (PKA) protocol $(PKA\_Setup, PKA\_Prove, PKA\_Verify)$ is said to be ε-secure against* selective forgery under an adaptive chosen message attack *if every polynomial time probabilistic forging algorithm B playing against a challenger wins the following game with probability at most ε:*

1) *The forger B picks a target message m and sends it to the challenger.*
2) *The challenger runs the setup algorithm for the PKA, sends PK to the forger B and keeps SK secret.*
3) *The forger B mounts an adaptive chosen message attack: B selects messages $m_1, \ldots, m_\ell$, where $\forall i$: $m_i \neq m$; for each m_i, the challenger (acting as prover) and B (acting as verifier) engage in an authentication session.*
4) *At some point of the forger B's choosing, B (acting as prover) attempts to authenticate the target message m to the challenger (acting as verifier). The authentication sessions for the m_i's may be running concurrently.*

We say that the forger B wins the game if the challenger accepts the authentication on m.

Definition IV.2. ***Existentially-secure PKA.*** *A PKA protocol $(PKA\_Setup, PKA\_Prove, PKA\_Verify)$ is said to be ε-secure against* existential forgery under an adaptive chosen message attack *if the same conditions as in Definition IV.1 hold, except that the forger B can pick the target message m at any point in the game.*

From PSR to Selectively-secure PKA. We construct a selectively-secure PKA protocol from a PSR system $(PSR\_Setup, PSR\_Query, PSR\_Answer, PSR\_Verify)$ that is selectively secure against polynomial-time adversaries (per Definition II.5) as follows:

PKA Setup. Select a uniformly random message $m^* \in U$, define $R = \{m^*\}$ and denote $v(\cdot)$ as the function that returns 1 on m^* and $\perp$ otherwise. Run the PSR setup algorithm $PSR\_Setup(R, v(\cdot), 1^k)$, and return (PK, I_S) as the public and secret keys for the PKA.

PKA Prover. The PKA prover has the public and secret keys (PK, I_S) for the PKA, and must authenticate message m_i. To do this, the PKA prover acts as the *secondary* in the PSR system, and proves that $m_i \notin R$; that is, the PKA prover receives q, and returns the output of $PSR\_Answer(q, I_S, PK)$.

PKA Verifier. The PKA verifier has the public key PK, and acts as the *resolver* in the PSR system. The PKA verifier runs $PSR\_Query(x, PK)$ to obtain q and sends q to the PKA prover; upon receiving the response, the PKA verifier accepts if $PSR\_Verify(b, v, \pi)$ accepts.

Remark IV.3. Note that our PKA construction does not satisfy perfect completeness (if the verifier happens to choose m^*, the nameserver cannot authenticate that message). If the PSR system has perfect completeness (note: NSEC5 doesn't have prefect completeness due to hash collisions), then we can also get a PKA system with perfect completeness. To do this, we add a bit to each element in the universe, indicating whether it is a real or dummy element. We choose the message m^* to be a dummy element, by prepending a 0 as its first bit. Then, when a resolver wants to query on message m, it asks for $1m$ instead; in this way, all messages in the universe can be authenticated. Thus, if the PSR system has perfect completeness, then the constructed selectively-secure PKA does as well.

Theorem IV.4. *Suppose we have a PSR system that is ε-secure against* selective membership under an adaptive-chosen-message attack *and μ_S-secure against soundness attackers. The PKA scheme constructed above is ε'-secure against* selective forgery under an adaptive-chosen-message attack*, where $\varepsilon' = 2\varepsilon + \mu_s$*

This theorem also holds with a notion of PSR soundness that is weaker than that of Definition II.3; instead of requiring soundness even in the face of compromised nameservers, with *weak soundness* we trust secondary nameservers and only require that other network adversaries cannot produce false DNS responses. The weak soundness adversary has knowledge of PK but not I_S, and communicates concurrently with both a secondary and a resolver, while trying to prove a false statement which will be accepted by the resolver. We denote $A^{Answer_{(I_S,PK)}}$ as an adversary A with oracle access to an honest secondary nameserver.

Definition IV.5. ***Weak Soundness:*** *for all probabilistic polynomial time stateful adversaries A we have*

$$\Pr\left[\begin{array}{l} (R, v(\cdot)) \overset{R}{\leftarrow} A(1^k); \\ (PK, I_S) \overset{R}{\leftarrow} Setup(R, v(\cdot), 1^k); \\ x \overset{R}{\leftarrow} A^{Answer_{(I_S,PK)}}(PK); \\ q \overset{R}{\leftarrow} Query(x, PK); \\ (b', v', \pi) \overset{R}{\leftarrow} A(PK) : \\ Verify(b', v', \pi) = 1 \ \wedge \\ ((x \in R \ \wedge \ (b' = \text{'}no\text{'} \ \vee \ v' \neq v(x))) \ \vee \\ (x \notin R \ \wedge \ b' = \text{'}yes\text{'})) \end{array}\right] \leq \mu(k)$$

for a negligible function $\mu(k)$.

Proof of Theorem IV.4: Suppose that there exists a polynomial-time PKA forger B which manages to win the selective forgery security game (Definition IV.1) for the constructed PKA scheme with non-negligible probability ε'. We describe a polynomial-time PSR adversary A that uses the PKA forger B as a subroutine to win the PSR selective-membership security game (Definition II.5) with a non-negligible advantage $\varepsilon = \frac{\varepsilon'}{2} - \frac{\mu_s}{2}$. The PSR adversary A is as follows:

1) The PSR adversary A starts by obtaining the target message m that is output by the PKA forger B. A then draws a random message $m^* \neq m$ and sends the PSR challenger two target elements m, m^*, an empty target set $S = \phi$, and a value function v for the elements in $S \bigcup \{m, m^*\}$ where $v(m) = v(m^*) = 1$ and $\perp$ otherwise.
2) The PSR challenger selects either $R = \{m\}$ or $R = \{m^*\}$, each with probability $\frac{1}{2}$. Next, the PSR challenger runs $PSR\_Setup(R, v(\cdot), 1^k)$, sends the output PK to the PSR adversary A, and keeps I_S secret.

3) The PSR adversary A now emulates the PKA forger B as it mounts its adaptive chosen message attack. A relays each message $m_1, \ldots, m_\ell$ sent by the PKA forger B for authentication to the (PSR) challenger; A then relays the answers produced by the challenger back to the PKA forger B.
4) At this point, the PSR adversary A needs to output a bit g, with $g = 0$ if A believes that $m \in R$ and $g = 1$ if A believes $m^* \in R$. To do this, A waits until the PKA forger B is ready to forge an authentication for its target message m; when this happens, the PKA forger B (acting as prover) authenticates m to the PSR adversary A (acting as verifier). If A accepts the authentication, it returns $g = 1$; otherwise, it returns $g = 0$.

To analyze the success probability of the above PSR adversary A we have two cases: In the first case, $R = \{m^*\}$ and the PKA adversary B has to forge a proof that target message m is such that $m \notin R$; thus, in this case the PKA adversary B witnesses the exact same view as in a real execution. Thus, with probability at least ε', the PKA forger B wins his game, and A correctly identifies that B succeeded in forging the authentication on message m. So the probability A wins in this case is at least ε' (by correctly guessing $g = 1$).

In the second case, $R = \{m\}$ and the PKA adversary B has to forge a proof that target message m is such that $m \notin R$; in this case it is no longer true that the PKA adversary B witnesses the exact same view as in a real execution. Indeed, the PKA adversary B is now asked to prove a *false* statement (namely, that $m \notin R$ where $R = \{m\}$), so the PSR's *soundness* property, through a straightforward reduction, ensures that B cannot generate a proof for a false statement with probability larger than μ_s. (Note that the full-fledged soundness property of PSR is not needed here—it suffices for PSR to be sound against adversaries who can issue queries, but do not have the data I_S.) Thus, the probability that the PSR adversary A wins (by correctly guessing the bit $g = 0$) is at least $(1 - \mu_s)$.

Since both cases are equally likely, the PSR adversary A wins the game with probability at least $\frac{1}{2}(\varepsilon') + \frac{1}{2}(1 - \mu_s) = \frac{1}{2} + \frac{\varepsilon'}{2} - \frac{\mu_s}{2}$. ■

From selective to existential security. Now, we show that in the random oracle model, a PKA that is selectively secure (Definition IV.1) can be used to construct a PKA scheme which is existentially secure (Definition IV.2). In combination with Theorem IV.4, this shows that a PSR system implies a strong notion of security for a PKA scheme.

To do this, we simply use the random oracle to hash each PKA message before we authenticate it (and modify the other PKA algorithms appropriately). Call this scheme "hashed PKA". The running time of the resulting PKA prover will be greater than the running time of the secondary nameserver in the PSR by just one random-oracle query.

Theorem IV.6. *Suppose that we have a PKA scheme* $(Setup, Prove, Verify)$ *which is* ε'*-secure against* selective forgery under an adaptive-chosen-message attack. *Then, in the random oracle model, the "hashed PKA" scheme is* ε*-secure against* existential forgery under an adaptive-chosen-message attack*, where* $\varepsilon' = \varepsilon / q(k)$ *and* q *is some polynomial in* k.

Proof: Suppose that there is an adversary E which wins the existential-security game for the "hashed PKA" (per Definition IV.2) in the random-oracle model with non-negligible probability ε. We describe an adversary B that uses E as a subroutine to win the selective-membership security game for the PKA (Definition IV.1). In the random oracle model (per [16], [36]), we 'program' the random oracle h, and give the adversary E the value of $h(m)$ each time the random oracle is called on some some message m. Since adversary E runs in polynomial time, we know E can make at most a polynomial number of queries $q(k)$ to the random oracle. The selective-security adversary B is as follows:

1) Selective-security adversary B picks a uniformly-random message m; it sends m as its target message to the challenger. B also draws a random $j \in \{1, \ldots, q(k)\}$.
2) The challenger runs the setup algorithm for the PKA protocol, sends PK to the selective-security adversary B, and keeps SK secret.
3) B emulates E as it mounts its adaptive chosen message attack, by functioning as an intermediary between the challenger and E and relaying their authentication messages to each other. When E queries the random oracle h, B answers with random values at all queries except the j^{th} one. When E sends its j^{th} random oracle query for some message m_j, then B sets $h(m_j)$ equal to m; if E asks to authenticate the message m_j before E's forgery attempt, then B fails.
4) Finally E tries to forge an authentication for some message m^*. If $h(m^*) = m$, then B uses this forged authentication to authenticate his target message m. Otherwise, B fails.

We may assume that E accesses h on the message m^* that B tries to forge (otherwise it has negligible success probability). It follows that adversary B sets $h(m^*) = m$ with probability $\frac{1}{q(k)}$. Thus, B wins its game if (1) E managed to successfully forge an authentication for his target message, and (2) B picks the correct j, which occurs with probability $\varepsilon' = \frac{\varepsilon}{q(k)}$. ■

B. Digital Signatures from PSR with Simple Query Algorithms

We have seen that PSR systems can be used to construct public-key authentication schemes of the same computational complexity. We now point out that PSR systems that satisfy some restrictions on their query algorithm can actually be used to construct signature schemes of about the same complexity. This shows that secondary nameservers in any PSR (whose query algorithm satisfies these restrictions) must *inherently* perform online public-key operations.

Consider any PSR system where the $Query$ algorithm is deterministic (as in our NSEC5 construction). Now apply the transformation of Section IV-A to obtain a PKA scheme. Observe that interaction is not necessary in the resulting PKA scheme, because the PKA prover can compute the PKA verifier's first message on its own. Thus, the resulting PKA scheme is actually a signature scheme. The complexity of the signer is the same as the complexity of the $Query$ and $Answer$ algorithms in the PSR schemes.

Now suppose the PSR $Query$ algorithm is randomized. The transformation of Section IV-A above does not work, because the PKA prover cannot be trusted to choose, or even to know, the randomness that the $Query$ algorithm uses. We get around this problem of choosing randomness by using the approach of Fiat and Shamir [21]: namely, apply the transformation of Section IV-A, but let the randomness for the $Query$ algorithm be $h(x)$, where h is a random oracle. This allows the PKA prover to know the randomness and thus to compute the PKA verifier's first message q. However, now the security proof for the resulting signature scheme works only if soundness for the PSR schemes holds against an adversary who knows the randomness used in the $Query$ algorithm.

Thus, we obtain the following theorem.

Theorem IV.7. *Consider any PSR system* $(Setup, Query, Answer, Verify)$ *for which the Query algorithm is deterministic, or for which the randomness of the Query can be given to the adversary without harming soundness. Such a system implies an existentially-unforgeable digital-signature scheme with signing complexity equal to the complexity of* $Query$ *plus* $Answer$ *(plus at most two random oracle queries).*

Other PSR models? Although our PSR model separates nameservers into two parties (primary and secondary), we still obtain a signature scheme even without this separation *i.e.,* if we have just a primary nameserver that commits to R and $v(\cdot)$ and *honest* secondaries, or when the trusted primary nameserver itself responds directly to resolvers. This means that even without the separation of nameservers into trusted primary and untrusted secondary, a nameserver still have to generate signatures online in order to both prevent zone enumeration and maintain soundness.

C. Weaker notions of privacy?

We have shown that we can use a PSR system satisfying both weak soundness, and our definition of privacy against zone enumeration, namely the selective membership requirement (Definition II.5), in order to build signatures and public-key authentication (PKA) schemes. We therefore want to claim that we demonstrated that the work involved in a secure PSR must be non-trivial, thus explaining why a protocol like NSEC3, which only uses hashing, inherently cannot prevent zone enumeration. However, one could protest and argue that our privacy definitions are too strong, and that it may be possible to have a more relaxed definition of privacy that still prevents zone enumeration. We argue that this is not the case.

Suppose we modify the f-zk privacy definition while still protecting against a resolver that produces an answer for an element that it did not explicitly query on (*i.e.,* the essence of zone enumeration). A little more formally, suppose that there is some distribution on the set R. We require that for every probabilistic polynomial time adversary A, there exists a simulator A' with oracle access to the set R, such that if A interacts with a PSR system as a resolver and outputs, at the end of the interaction, an element x he believes to be in the set R which A has not explicitly queried (this is 'success' if $x \in R$), there is a simulator A' that interacts with an oracle to the set R, which is successful as well with similar probability. (Here similar means that the difference is negligible.) We can show that under this requirement we may obtain a notion related to selective membership, where instead of two elements chosen by the adversary, the two elements of the challenge are chosen at random, by applying a similar reduction to the one in Theorem II.6. We can also show that the latter implies public-key *identification*, under a similar reduction to Section IV-A. Therefore we claim that we have demonstrated that preventing zone enumeration requires non-trivial computation.

V. Further Work

In a companion paper [35] we generalize the constructions of this paper and show how to obtain PSR systems without random oracles. In [35] we suggest a general construction based on a weaker variant of verifiable random/unpredictable functions [32]. We describe a construction based on *hierarchical identity based encryption*, and in particular the one by Boneh, Boyen and Goh [15], which does not reveal any information about the set R, not even its cardinality. Lastly we present an interesting construction which uses the unique structure of cuckoo hashing [38] to construct a PSR that is based on factoring and the discrete logarithm assumption.

We are also working towards designing and testing practical implementations of NSEC5.

VI. Acknowledgements

We thank Casey Deccio, Paul Hoffman, Daniel Kahn Gillmor, Ben Laurie, Ondrej Sury, Jared Mauch, Matthijs Mekking, Benno Overeinder, Jan Vcelak, Wouter Wijngaards, various other attendees of IETF'90, IETF'91 and the DNS OARC Fall 2014 workshop, and the anonymous NDSS'15 reviewers for useful discussions and comments.

This material is based upon work supported by the US National Science Foundation under Grants 017907, 1347525, 1012798, and 1012910, the Israel Science Foundation, BSF and Israeli Ministry of Science and Technology, and from the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation.

References

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

[2] P. Albitz and C. Liu, *DNS and Bind.* O'Reilly Media, Inc., 2001.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," IETF, RFC 4033, Mar. 2005.

[4] ——, "Protocol Modifications for the DNS Security Extensions," IETF, RFC 4035, Mar. 2005.

[5] ——, "Resource Records for the DNS Security Extensions," IETF, RFC 4034, Mar. 2005.

[6] R. Arends, M. Kosters, and D. Blacka, "DNS Security (DNSSEC) Opt-In," IETF, RFC 4956, Jul. 2007.

[7] D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS)," IETF, RFC 3833, Aug. 2004.

[8] J. S. B. Kaliski, *RFC 2437: PKCS #1: RSA Cryptography Specifications, Version 2.0.* Internet Engineering Task Force (IETF), 1998.

[9] J. Bau and J. C. Mitchell, "A security evaluation of dnssec with nsec3," in *NDSS*, 2010.

[10] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS*, 1993.

[11] ——, "Optimal asymmetric encryption," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 950. Springer, 1994, pp. 92–111.

[12] ——, "The exact security of digital signatures — how to sign with RSA and Rabin," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 1070. Springer, 1996, pp. 399–416.

[13] D. J. Bernstein, "Nsec3 walker," http://dnscurve.org/nsec3walker.html, 2011.

[14] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," *Algorithmica*, vol. 12, no. 2/3, 1994.

[15] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *EUROCRYPT*, 2005.

[16] R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited," *J. ACM*, vol. 51, no. 4, pp. 557–594, 2004.

[17] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin, and L. Reyzin, "Mercurial commitments with applications to zero-knowledge sets," in *EUROCRYPT*. Springer, 2005.

[18] J. Coron, "On the exact security of full domain hash," in *CRYPTO*, 2000.

[19] D. Dolev, C. Dwork, and M. Naor, "Nonmalleable cryptography," *SIAM J. Comput.*, vol. 30, no. 2, pp. 391–437, 2000.

[20] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *STOC*. ACM, 1987, pp. 210–217.

[21] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.

[22] E. Ghosh, O. Ohrimenko, and R. Tamassia, "Verifiable order queries and order statistics on a list in zero-knowledge," *IACR Cryptology ePrint Archive*, no. 2014/632, 2014.

[23] R. Gieben and W. Mekking, "DNS Security (DNSSEC) Authenticated Denial of Existence," IETF DNSEXT Internet Draft http://tools.ietf.org/html/draft-gieben-nsec4-00, January 2012.

[24] ——, "Authenticated Denial of Existence in the DNS," IETF, RFC 7129, Feb. 2014.

[25] O. Goldreich, *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[26] ——, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[27] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.

[28] D. Kaminsky, "Phreebird," http://dankaminsky.com/phreebird/, 2011.

[29] O. Kolkman, W. Mekking, and R. Gieben, "DNSSEC Operational Practices, Version 2," IETF, RFC 6781, Dec. 2012.

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

[31] S. Micali, M. O. Rabin, and J. Kilian, "Zero-knowledge sets," in *FOCS*. IEEE Computer Society, 2003, pp. 80–91.

[32] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," in *FOCS*. IEEE Computer Society, 1999, pp. 120–130.

[33] A. Miller, M. Hicks, J. Katz, and E. Shi, "Authenticated data structures, generically," in *POPL*. ACM, 2014.

[34] M. Naor and K. Nissim, "Certificate revocation and certificate update," *IEEE J. on Selected Areas in Communications*, vol. 18, no. 4, 2000.

[35] M. Naor and A. Ziv, "Primary-secondary-resolver membership proof systems," *IACR Cryptology ePrint Archive*, no. 2014/905, 2014.

[36] J. B. Nielsen, "Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case," in *CRYPTO*. Springer, 2002.

[37] R. Ostrovsky, C. Rackoff, and A. Smith, "Efficient consistency proofs for generalized queries on a committed database," in *ICALP*, 2004.

[38] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *Algorithms - ESA*, 2001.

[39] PowerDNS, *PowerDNS manual*, December 2013.

[40] V. Ramasubramanian and E. G. Sirer, "Perils of transitive trust in the domain name system," in *ACM IMC*, 2005.

[41] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

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

[43] V. Shoup, "OAEP Reconsidered," in *CRYPTO*, 2001, pp. 239–259.

[44] G. Sisson and B. Laurie, "Derivation of DNS Name Predecessor and Successor," IETF, RFC 4471, Sep. 2006.

[45] R. Tamassia and N. Triandopoulos, "Certification and authentication of data structures," in *AMW*, 2010.

[46] M. Wander, L. Schwittmann, C. Boelmann, and T. Weis, "GPU-Based NSEC3 Hash Breaking," in *IEEE Symp. Network Computing and Applications (NCA)*, 2014.

[47] S. Weiler and J. Ihren, "Minimally Covering NSEC Records and DNSSEC On-line Signing," IETF, RFC 4470, Apr. 2006.

Appendix

A. The Random Oracle model

In Section III-C we proved the security of NSEC5 in the random oracle model, per the model in [16]. The random oracle model has been used quite extensively to analyze cryptographic protocols [10]–[12], [18], [43]. In the random oracle model, all parties, including adversaries, can query a random oracle O. According to the security parameter k and a length function $\ell_{out}(\cdot)$, the oracle O is a function chosen uniformly at random out of all possible functions mapping $\{0,1\}^*$ to $\{0,1\}^{\ell_{out}(k)}$. A system is considered secure when its adversary has a negligible probability of success or a negligible advantage at winning the security game, but the probability is also taken over the choices of the random oracle. In the security proof, the random oracles can be "programmed", *i.e.,* certain answers to random oracle queries can be set either beforehand or on the fly to be specific values by a simulator (see Nielsen [36]), with the restriction that the answers must look random and be consistent (*i.e.,* each time O is queried on x during the simulation, it returns the same value).

B. Signature schemes

We define public key signature schemes as in [26].

Definition A.1. *A signature scheme is defined by three (polynomial time) algorithms (G, S, V): The generator G gets the security parameter k and outputs two keys, a signing key SK and a verification key PK, $G(1^k) = (SK, PK)$. The signing algorithm S takes the secret key SK and a message $m \in \{0,1\}^\ell$ and produces a signature. The verification algorithm V gets PK and a presumed signature to a message and verifies it,* i.e., *outputs 'accept' ('1') or 'reject' ('0'). We require* perfect completeness*: For every pair of keys (SK, PK) generated by $G(1^k)$ and for every message $M \in \{0,1\}^{\ell = p(k)}$ (every message of length at most polynomial in the security parameter) it holds that*

$$Pr[V_{PK}(S_{SK}(m), m) = 1] = 1$$

Our security requirement is "existential unforgeability against chosen message attacks", *i.e.,* that even an adversary who obtains a polynomial number of signatures for messages of his choosing, still cannot generate a valid signature on a new message for which he did not obtain the signature [27].

Definition A.2. *A signature scheme is (t, ε) existentially secure against chosen message attacks if every probabilistic adversary A (whose running time plus description size are at most t) wins the following security game, played against the challenger, with probability at most ε.*

- *The challenger runs the setup algorithm $S(1^k)$ and obtains (SK, PK), sends PK to the adversary and keeps SK secret to himself.*
- *The adversary A issues an adaptively chosen sequence of messages $m_1, .., m_q$ to the challenger and gets a signature on each of those messages $s_1, .., s_q$ where $s_i = S_{SK}(m_i)$. By adaptively chosen we mean that the adversary chooses m_{i+1} only after seeing signature s_i.*
- *The adversary chooses a message m^* together with a forged signature s^* and sends them to the challenger; The only restriction is that $m^* \neq m_i$ for every i.*

The adversary wins the game when $V_{PK}(s^*, m^*) = 1$, i.e., *the forged signature is accepted as valid.*

If for every polynomial $t(k)$ there exists a negligible function $\varepsilon(k)$, we call the scheme existentially secure against chosen message attacks.

C. RSA and Trapdoor Permutations

The RSA trapdoor permutation [41] has three algorithms. The key generator G gets the security parameter k and outputs two keys, a public key $PK = (N, e)$ and private key $SK = (N, d)$ as follows: (1) The exponent e can be a small deterministic number (say 3); a small e means that the RSA forward algorithm will be fast. (2) The RSA modulus N is chosen as $N = P \cdot Q$ where P and Q two random primes such that e is relatively prime to $P - 1$ and to $Q - 1$ and $N = PQ$ is of length k bits. (3) d is chosen such that $d \cdot e \equiv 1 \bmod \operatorname{lcm}(P-1, Q-1)$. The RSA forward algorithm takes a value $x \in \mathbb{Z}_N$ and the public key PK and computes $y = x^e \bmod N$. The RSA inverse algorithm takes a value $y \in \mathbb{Z}_N$ and the secret key SK and computes $x = y^d \bmod N$. We use the following properties of RSA.

RSA is a permutation over all of $\mathbb{Z}_N$. Every value $x \in \mathbb{Z}_N$ is mapped by the RSA forward algorithm to some unique $y \in \mathbb{Z}_N$; the RSA inverse algorithm maps y back to x (note that this holds even when x is not relatively prime to N).

The RSA hardness assumption with respect to exponent e and security parameter k. The (t, ε) RSA assumption states that for any adversary whose description size and running time add up to t, for a random k-bit product N of two primes P, Q such that e is relatively prime to $P-1$ and $Q-1$, for a random $y \in \mathbb{Z}_N$, $\Pr[A(y, N, e) = x \textit{ and } x^e \equiv y \bmod N] \leq \varepsilon$. (The asymptotic form of this assumption states that there exists a negligible function $\varepsilon(k)$ for every polynomial $t(k)$.)

RSA random self-reducibility Succeeding in finding the RSA inverse of any element of a set of r random challenges is just as hard as succeeding in finding the inverse of a single fixed challenge. Specifically, under $(t, \varepsilon - r/N)$ RSA assumption, no adversary running in time $t' \approx t$ can find the RSA inverse of any element in a set of r random challenges in Z_N with probability greater than ε. This follows because if we have an algorithm A that finds the RSA inverse of some element in a set of r random challenges, then given a single challenge z, we can find its inverse is follows. In the unlikely case $z = 0$, then the inverse is 0; else, in the unlikely case $\gcd(z, N) > 1$, finding the inverse is easy since $\gcd(z, N)$ gives us factorization of N. Else (in the likely case), select random $w_i \in \mathbb{Z}_N$ for $1 \leq i \leq r$, generate $z_i = z \cdot w_i^e \bmod N$, and run A on the set $\{z_i\}_{i=1}^r$ (note that this set has r random elements in Z_N since z is coprime with N). Assume no $w_i = 0$ (else, abort; this will happen with probability at most r/N). Upon obtaining the RSA inverse of some z_i, divide the result by w_i to get the RSA inverse of z (division will fail in the unlikely case when $\gcd(w_i, N) > 1$, but if it fails, then $\gcd(w_i, N)$ will give us the factorization of N and will allow us to invert RSA on z, anyway, since $w_i \neq 0$).