NSEC5: Provably Preventing DNSSEC Zone Enumeration

Sharon Goldberg*, Moni Naor†, Dimitrios Papadopoulos*
Leonid Reyzin*, Sachin Vasant*, Asaf Ziv†

***Abstract*—We use cryptographic techniques to study zone enumeration in DNSSEC. DNSSEC is designed to prevent network 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 network attackers and privacy against zone enumeration cannot be satisfied simultaneously unless the DNSSEC server performs online public-key cryptographic operations.

We then propose a new cryptographic construction that solves the problem of DNSSEC zone enumeration while remaining faithful to the operational realities of DNSSEC. 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 only harms privacy against zone enumeration, by effectively downgrading the security of NSEC5 back to that of 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 an authoritative nameserver, rather than an arbitrary network 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 multiple times. One of the most interesting of these issues is the problem of *zone enumeration* [Ber11], [BM10], [AL01]. 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 [WI06], RFC 5155 [LSAB08]) 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 previous techniques based on hashing failed to solve the problem. Our result shows that achieving privacy guarantees in this setting (while preserving the security property of DNSSEC) necessitates the use of public-key cryptographic operations in the online phase of the protocol. Moreover, we provide a new cryptographic construction that addresses the problem of DNSSEC zone enumeration while remaining faithful to the operational realities of DNSSEC.

*Boston University, Department of Computer Science. Email: {goldbe,dipapado,reyzin,sachinv}@cs.bu.edu. Research supported in part by the US National Science Foundation under grants 1017907, 1347525, 1012798, and 1012910.

†Weizmann Institute of Science, Department of Computer Science and Applied Mathematics. Email:{moni.naor,asaf.ziv}@weizmann.ac.il. Incumbent of the Judith Kleeman Professorial Chair. Research supported in part by grants from 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.

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 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 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 [AAL+05c] 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 DNSSEC community initially arguing that it is actually *not* a problem (*e.g.,* RFC 4033 [AAL+05a]), before eventually arriving at consensus that it is a problem for some zones (RFC 5155 [LSAB08]). 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:

> Though the NSEC RR meets the requirements for authenticated denial of existence, it 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 [San04], [Ait11, pg. 37].

Thus, in 2008, RFC 5155 [LSAB08] suggested NSEC3, a precomputed denial of existence technique, designed to make zone enumeration more difficult. With NSEC3, first 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]

Hashing the names makes trivial enumeration of the zone much more difficult, but the design nevertheless remains vulnerable to zone enumeration using an offline dictionary attack. Specifically, an adversary can issue several queries for random non-existent names, obtain a number of NSEC3 records, and then use rainbow tables (or other dictionary attacks for cracking hashes) to determine the names that are present in the zone from the hashes in the NSEC3 records. Indeed, Bernstein's nsec3walker tool [Ber11] 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 [WSBW14] used a GPU to reverse 64% of the NSEC3 hashes in the .com zone in 4.5 days. Indeed, RFC 5155 (Sec. 12.1.1) acknowledges these zone enumeration attacks.

To blunt the impact of dictionary attacks, the RFCs do introduce a salt value (using the NSEC3PARAM record); however, in contrast to password-hashing applications that mitigate against dictionary attacks by using a *unique* salt for each user, RFC 5155 requires 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 [KMG12] 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

Our story thus begins here. 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

[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`) [RS05].

[2]For simplicity of exposition, we ignore the issues of wildcard records and enclosers in our descriptions of NSEC and NSEC3; see RFC 7129 [GM14].

[3]There was also an Internet Draft [GM12] (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.

addressed by previous designs, and propose a new solution, called NSEC5, that protects against them.

Our first contribution is the following cryptographic model, which makes precise the desired notion of privacy:

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*.

Our goal is to design a denial-of-existence mechanism that achieves the following:

(1) Soundness. The primary nameserver is trusted to determine the set R of names in the zone, and to provide correct responses to DNS queries. However, the secondary nameservers and other network adversaries are not trusted to provide correct responses to DNS queries. The soundness property 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" described in RFC 3833 [AA04].

(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 the formalization of this requirement to avoid zone enumeration, raised, *e.g.,* in RFC 5155 [LSAB08]

(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 [WI06].

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

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

Moreover, we 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 interactive message authentication protocol, for which the best known solution is a cryptographic signature anyway. In Section IV-C we discuss the question of whether our privacy requirements are "too strong" and argue that any meaningful relaxation still implies public-key authentication. Thus we conclude that preventing zone enumeration requires substantial ("public-key") online computation, rather than just private-key computation such as evaluating a cryptographic hash function (as in NSEC3).

E. NSEC5: A Denial-of-existence Mechanism

Armed with the knowledge that privacy necessitates an online signature computation for every negative response, we present a new solution that requires two online hash computations and a single online RSA computation for each authenticated denial of existence. Our solution, called NSEC5, provably achieves soundness and privacy.

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 DNSSEC with NSEC and NSEC3, and allow only the primary nameserver to know the primary secret key SK_P for the zone; this primary secret key is used to ensure 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 we provide 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: all we need to do is 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 hash function h_1 (modeled as random oracle [BR93])

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

and hashes it to a short string with another hash function h_2 (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 (just like 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 $S(q)$ and $F(q)$ using SK_S, and responds to the resolver with (1) an NSEC5PROOF record containing the value $S(q)$ and (2) the signed NSEC5 record for the hashes that are lexicographically before and after $F(q)$.

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 verify $S(q)$ in the NSEC5PROOF, checking that

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

and (3) checking that $h_2(S(q))$ (from the NSEC5PROOF) is lexicographically between the hashes in the NSEC5 record. Thus, $S(q)$ maintains soundness by acting as a "proof" that the value $F(q)$ 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 formally prove that our construction satisfies both 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!).

Soundness and secret key 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.

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, (s)he would not be able to produce a properly-signed NSEC5 record covering $F(q)$, even if (s)he knew the secret secondary key SK_S.

Performance. Our solution allows resolvers to verify using the same technologies they always used: hashing and validation of RSA signatures. NSEC5 does, however, require a single online RSA computation at the secondary nameserver, making it more computationally heavy than NSEC and NSEC3 (and NSEC4). However, our lower bounds do prove this extra computation is necessary to eliminate zone enumeration. Additionally, only the zone administrators that require our strong privacy guarantees need to deploy NSEC5; others that don't can just use NSEC or NSEC3. We discuss other practical issues regarding NSEC5 deployment in Section III-B.

Indeed, online signing for denial of existence was already proposed in RFC 4470 [WI06], further discussed in RFC 4471 [SL06], and implemented in nameserver software like powerDNS [Pow13, Sec. 4] and Phreebird [Kam11]. These 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". Some criticized the RFC 4470 solution because it compromises soundness (if a secondary nameserver is hacked or leaks its key). In contrast, our solution has the same computational complexity without the same risks to soundness, because the online signing is used only for looking up the correct NSEC5 record and cannot be used to produce a false denial-of-existence response. Thus, NSEC5 preserves soundness even when the secondary nameserver is compromised, or its secret secondary key SK_S is leaked.

We therefore believe NSEC5 presents an attractive alternative to NSEC3 for those zone operators who require strong privacy against zone enumeration. Moreover, because NSEC5 is structurally very similar to NSEC3, it can incorporate the other performance and policy optimizations developed for DNSSEC, including NSEC3 opt-out or the space-saving techniques proposed in NSEC4 [GM12].

F. Organization & Contributions

The organization of this paper follows the summary above. Section II presents our model and security definitions; we use a traditional DNSSEC notion of soundness, and our main definitional contribution is in our notion of privacy. Our next contribution is our NSEC5 construction; we present NSEC5 in Section III and prove it satisfies soundness and privacy. Our final contribution is a number of cryptographic lower bounds, which explain why NSEC5 requires online signing at the secondary nameserver in order achieve simultaneous soundness and privacy, which we present in Section IV. Our results are supported by the standard cryptographic definitions (signatures, random oracles) in Appendix A.

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*, introduced by Micali, Rabin and Kilian (ZKS for short) and its generalization to zero-knowledge elementary databases [MRK03]. The latter is 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 [CHL+05] 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 and by working in this less stringent setting, we are able to obtain more efficient constructions.

Data structures that come with soundness guarantees are also relevant (see e.g. [BEG+94], [NN00], [TT10], [MHKS14]).

[4] [CHL+05] requires the verifier to verify $\log |U|$ mercurial commitments, where U is the universe of elements and each verification involves a "public-key operation".

These data structures return an answer along with a proof that the answer is sound; "soundness" means that the answer is consistent with some external information. We also need soundness in our setting, but we augment this with the additional requirement of privacy against zone enumeration.

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). 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 privileged 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 it does not change it 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. We do allow a *negligible* probability of failure.

Definition II.2. ***Completeness:*** *For all $R \subseteq U$ and for all $v : R \to V$ and $\forall x \in U$,*

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

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.,* [NN00]) but we chose not to concentrate on this here. 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*

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

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 [Gol01, 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"[7]. 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:

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

or by the simulator that generates fake parameters:

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

Next, the adversary starts issuing queries q_i (adaptively), based on the public parameters and previous responses to queries it got. 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 it was given.

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 $v : R \to V$ the distinguisher D cannot distinguish between the following two views:

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

and

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

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).

[7] Referring to a resolver as an "adversary" when studying f-zero knowledge makes sense, since the resolver is the adversary that wishes to break the privacy of the system.

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 [Gol01, 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 prob-

S_{rsa}, RSA, RSA^{-1}	RSA algorithms
PK_S, SK_S	RSA keys (secondary keys)
S_{sig}, Sig, Ver	Signature scheme algorithms
PK_P, SK_P	Signature scheme keys (primary keys)
h_1	Random oracle from U to $\mathbb{Z}_N$
h_2	Random oracle from $\mathbb{Z}_N$ to $\{0,1\}^n$
$F : U \to \{0,1\}^n$	The function $h_2(RSA^{-1}_{SK_S}(h_1(\cdot)))$
$S : U \to \mathbb{Z}_N$	The function $RSA^{-1}_{SK_S}(h_1(\cdot))$
$R = \{x_1, .., x_r\}$	Set of existent domain names
U	Universe of domain names
V	Universe of IP addresses
$v : R \to V$	Function mapping domain names to IP addresses

Fig. 1. Table of notation.

ability 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 $2 \cdot \mu_f$. ■

III. NSEC5 Construction and Proof

Our NSEC5 construction was described in Section I-E. Here we show why our NSEC5 construction is a secure PSR system, and prove its security in the random oracle model. Table 1 summarizes our notation. Section III-A maps our NSEC5 construction to our formal model of a PSR system, while Section III-B discusses its practical considerations, including computational requirements, and the DNSSEC record types it requires. Our proof of security is in Section III-C.

A. Formally Modeling NSEC5 as a PSR System

We specify our NSEC5 construction in detail, and map it to our formal 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, where

$$h_1 : U \to \mathbb{Z}_{N_S}$$

is a "full-domain hash" [BR93] whose output size is exactly the size of the RSA modulus N_S, and

$$h_2 : \mathbb{Z}_{N_S} \to \{0,1\}^n$$

is a standard cryptographic hash function (*e.g.,* SHA-256) whose output length is chosen to prevent birthday attacks. Our security proofs model both h_1 and h_2 as random oracles. Finally, we use any existentially-unforgeable signature scheme (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 starts by generating the public parameters: It generates a key pair (PK_P, SK_P) for the existentially-unforgeable signature (the "primary keys"), generates an RSA key pair (PK_S, SK_S) (the "secondary keys"), and finally selects the hash functions h_1 and h_2. 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

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

which is then hashed to a shorter string using h_2

$$y = F(x) = h_2(\pi) \tag{3}$$

The y values are lexicographically ordered as $y_1, \ldots, y_r$, and $y_0 = 0^n$ and $y_{r+1} = 1^n$ are added. For $j \in \{0, .., r\}$, 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 $j = 0...r$, 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

$$\textit{'yes'}, (q, v(q)), Sig(q, v(q))$$

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 by locating index j for which $y_j < y < y_{j+1}$, and returns

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

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

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

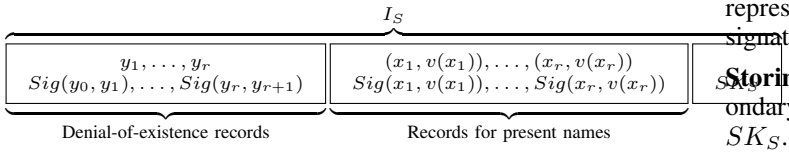

Fig. 2. Illustration of I_S.

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 [WI06]) and implemented in commercial DNSSEC systems [Pow13, Sec. 4], [Kam11].

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 the 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 *identical* computational overhead at the resolver. Specifically, RFC 5155 [LSAB08, Sec 10.3] specifies that the number of iterations in the NSEC3 hash can result in a computation with similar cost as verifying a signature on an NSEC3 record (*e.g.,* 500 SHA1 iterations for a 2048-bit RSA signature).

Finally, the primary 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.

Storing secrets at the nameservers. NSEC5 requires secondary nameservers to hold the the *secret* secondary key SK_S. 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). 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. This is in stark contrast to the RFC 4470 [WI06] online signing solution, that requires the secondary nameservers to hold the zone-signing key (*i.e.,* the primary secret key SK_P); with the RFC 4470 approach, soundness is completely lost if the secondary is compromised or the key is leaked.

Transmitting information to the resolvers. 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 $S(q)$. What DNSSEC record should be used to transmit $S(q)$? 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, called *e.g.,* 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. 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 [LSAB08]. 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] Alternatively, if we want to avoid including PK_S in each NSEC5 record (since an 2048-bit public RSA key is much larger than the 24-bit NSEC3 salt), we could use a separate DNSKEY record, setting the flag bits to indicate it is *not* a signing key for the zone (*e.g.,* using the reserved flag bits, or setting the ZSK and SEP bits to 0, or perhaps by introducing a new DNSKEY flag); this approach complies with RFC 4035 [AAL+05b]'s discussion on the use of the DNSKEY record for non-zone keys.

Transmitting information to secondary nameservers. Next, observe that the primary nameserver must communicate some extra information to the secondary nameserver during setup (apart from the usual A records, NSEC5 records, and RRSIGs); namely, the public parameters PK_S, h_1, h_2 and the secondary secret key SK_S. 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

[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.

an NSEC5PARAM records. Distributing the secondary *secret* keys SK_S may be more cumbersome, however, because secondaries traditionally do not store any secret keys in DNSSEC. Fortunately, since leaking SK_S does not compromise soundness in any way, we could just include SK_S in the NSEC5PARAM record (and require that NSEC5PARAM is never sent to resolvers), or alternatively have the primary directly send SK_S to the secondaries in a TSIG message[9].

Opt-out. Finally, the structural similarity of NSEC5 with NSEC3 and NSEC allows for easy adoption of existing mechanisms such as wildcards, Opt-out (RFC 5155) and Opt-in (RFC 4956 [AKB07]). We will address these complications in future versions of this work; here we only note that several space-saving proposals from the NSEC4 draft [GM12] can also be incorporated into our NSEC5 construction.[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.

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

Proof: We start by proving a few useful properties of the function F in equation (3). First, because h_1, h_2, and RSA are all deterministic algorithms and RSA is also a permutation, it follows that for every $x \in U$ there exists exactly one pair (y, π_y) for which it holds that (cf. equation (4))

$$F(x) = y, \quad h_2(\pi_y) = y, \text{ and } \quad h_1(x) = \pi_y^{e_S} \bmod N_S$$

The second property is verifiable pseudorandomness:

Lemma III.2. *For every $x \in U$, the value $F(x)$ is pseudo-random over $\{0,1\}^n$ in the following sense: no probabilistic polynomial-time adversary, who gets x and can ask for $F(x_i)$ and $S(x_i)$ on any sequence of points $x_1, x_2 \ldots$ not containing x, can distinguish $F(x)$ from a random value in $\{0,1\}^n$.* [11]

Proof: Assume to the contrary that there exists an adversary A which gets $x \in U$ and after the sequence of queries described, manages to distinguish $F(x)$ and a random value with a non-negligible advantage. We show that, using A, we can invert the RSA permutation with the same non-negligible probability, violating the RSA hardness assumption in Appendix C.

We are given a public RSA key (e, N) and challenge z that we wish to invert. Assume wlog that for every $x_i \neq x$ that A asks the random oracle to evaluate $h_1(x_i)$, it also asks to see $(F(x_i), S(x_i))$. Also assume wlog that the upper bound on the number of queries made by the adversary A is Q. Before A's first query, we draw uniformly at random $c_1, .., c_Q \in \mathbb{Z}_N$ (where $\mathbb{Z}_N$ is the domain/range of the RSA permutation) and use the public RSA key to compute $z_i = c_i^e \bmod N$. Now, every time A queries the h_1-random oracle on $x_i \neq x$ (that wasn't queried before), we answer with $h_1(x_i) = z_i$; we also return $S(x_i) = c_i$, return $F(x_i) = r_i$ where r_i is a random value, and remember that $r_i = h_2(c_i)$ is the output of the h_2-random oracle on input c_i. When h_1 is queried on x, we answer with the challenge z. When h_2 is queried on p we check if $z = p^e \bmod N$; if so, we have successfully inverted RSA on challenge z, and otherwise, we just answer in random and consistent (with previous answers) manner.

The distribution A witnesses is identical to the real distribution. There are two possible cases: If A did not query h_2 on p such that $z = p^e \bmod N$, it follows that A never learned the value of $F(x)$, and thus A cannot distinguish between $F(x)$ and a random string with greater than 0 advantage. Otherwise, we have successfully inverted RSA on challenge z. Thus, A's advantage in distinguishing $F(x)$ from a random string is the probability of successfully inverting RSA. ■

Corollary III.3. *The proof generalizes naturally to distinguishing a set of random values in $\{0,1\}^n$ from the true values of F on a set of elements $R \subset U$.*[12]

Corollary III.3 will helps us construct the simulator used to prove f-zk. We now prove Theorem III.1 by showing the following three properties (Definitions II.2, II.3 and II.4):

Completeness. For every $R \subseteq U$, $v : R \to V$ and $x \in U$, we need to show that that after we run $Setup(R, v(\cdot), 1^k)$ to get (PK, I_S) and $Answer(x, I_S, PK)$, the $Verify$ algorithm will output 1.

If $x \in R$, then from the definition of $Setup$, $Sig(x, v(x))$ will be a valid signature on $(x, v(x))$, and $Verify$ will output 1 with probability 1.

If $x \notin R$, we claim that $F(x) \neq y_j$ for every $j = 0...r$ with overwhelming probability. (y_j is defined in equations (2) and (3)). Otherwise, we could guess $F(x)$ with non-negligible probability without querying for $F(x)$, violating the pseudorandomness of F proved in Lemma III.2. Furthermore, an adversary cannot even find an element $x \notin R$ to violate the completeness property with non-negligible probability; if it could, this would again contradict Lemma III.2. To be exact, the probability for a collision for $x \notin R$ is at most $\frac{|R|}{|N_S|} + \frac{|R|}{2^n}$ (probability for a collision in h_1 plus that of h_2). Thus with Q attempts one would get a violation to the completeness requirement with negligible probability $\frac{Q \cdot |R|}{|N_S|} + \frac{Q \cdot |R|}{2^n}$.[13]

[9] TSIG is used by the primaries to authentically and dynamically update information stored at the secondaries.

[10] Using the wildcard optimization from NSEC4 [GM12], our denial-of-existence response (containing two NSEC5 records and two NSEC5PROOF records in the worst case) is only about one RSA-value (*e.g.,* 2048 bits) longer than today's unoptimized NSEC3 standard (containing three NSEC3 records in the worst case).

[11] Note that this means that the function $F()$ combined with $S()$ constitutes a selective VRF, as defined in [MRV99]. This is a very simple and efficient implementation of the primitive (albeit, only in the random oracle model).

[12] This follows because inverting RSA on any of a set of r values is as hard as inverting it on a single element (see Appendix C). This is the one place where we have used the 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.

[13] In the DNSSEC world, the set R changes dynamically, so if the adversary can affect the choice of the set R (after receiving the parameters (PK, I_S)), he could find a collision in Q attempts with probability $\frac{Q^2}{|N|} + \frac{Q^2}{2^n}$, due to the birthday paradox. Thus, we also require h_1, h_2 to be chosen so that this probability is negligible.

Soundness. The soundness of our proposal follows 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 algorithm A that breaks soundness with probability ε (per Definition II.3), we can construct a forging algorithm F that, given a public key PK_P, can win the existential unforgeability game with probability ε, where both algorithms have similar running times. Recall that in the existential unforgeability game, the forger F can ask a signing oracle for signatures on arbitrary messages and must forge a signature on a new message (Appendix B).

Initially the forger F receives the public key PK_P (and the security parameter k). It then starts running adversary A that breaks soundness as follows. First, the forger F 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 F runs the NSEC5 $Setup$ algorithm in Section III-A, with the following modifications:

- The forger F no longer generates the primary keys; instead, PK_P (that F was given as input) is used as the primary public key.
- The forger F does not have the private primary key SK_P. Instead, it queries its signature oracle to obtain signatures on pairs (y_j, y_{j+1}) for $j \in \{0, \ldots, |R|+1\}$ 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 $b' = 'no'$ 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 now show that the forger F did not request the signature σ when it ran its' modified NSEC5 $Setup$ algorithm; thus the forger F 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 F 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 $y_0, \ldots, y_{|R|+1}$ 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 F never queried its signing oracle for a signature on (y'_1, y'_2) during setup.

Suppose instead that $b' = 'yes'$ but $x \notin R$ or $v(x) \neq v'$. F can output the message (x, v') and its signature as the forgery; F wins the game because this signature was not requested by F during setup.

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

Privacy. In order to show that NSEC5 is f-zk for $f(R) = |R|$ we now construct a suitable simulator, where no probabilistic polynomial time distinguisher can distinguish between an interaction with the simulator and one with the real NSEC5 system. 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. SIM initializes by running the RSA setup algorithm to obtain secondary keys (PK_S, SK_S) and the signature scheme's setup algorithm to obtain primary keys (PK_P, SK_P). SIM then chooses the random oracles h_1, h_2. SIM then selects a list of $|R|$ uniformly random values in $\{0,1\}^n$, sorts them lexicographically to obtain $y_1, ..., y_{|R|}$, and adds the values $y_0 = 0^n$ and $y_{r+1} = 1^n$. For $j \in \{0, .., |R|\}$, each pair (y_j, y_{j+1}) is signed using the existentially-unforgeable signature algorithm and the primary secret key SK_P to obtain $Sig(y_j, y_{j+1})$.

SIM then outputs the fake public parameters $PK^* = (PK_S, PK_P, h_1, h_2)$ and fake secret information

$$SK^*_{SIM} = (SK_S, SK_P, \{y_j\}_{j=1}^{|R|}, \{Sig(y_j, y_{j+1})\}_{j=0}^{|R|},),$$

The secret simulator key (fake secret information) is very similar to the original secret information I_S that is usually given to the secondary nameserver after the $Setup$ algorithm in Section III-A terminates; the main difference is that instead of including the signatures $\{Sig(x, v(x))\}$ for $x \in R$, it contains the primary secret key SK_P instead. (This is natural, since when SIM initializes it does not know R.)

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 $x_i \notin R$ or $x_i \in R$ and its value v_i. If $x_i \in R$, SIM uses the secret key SK_P and the existentially-unforgeable signature scheme to produce a signature $Sig(x_i, v_i)$, and outputs

$$\textit{'yes'}, (x_i, v_i), Sig(x_i, v_i)$$

If $x_i \notin R$, then SIM uses SK_S to computes 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 through its secret parameters SK^*_{SIM} for an index j for which $y_j < y < y_{j+1}$. If such an index j is found, SIM returns

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

If no such index j is found, *i.e.,* a collision has occurred, SIM aborts, as it fails to produce a proof for $x_i \notin R$. Note this is exactly the case where our completeness requirement fails to hold (see our completeness argument earlier in this section), and so SIM, just like the real NSEC5 system, fails to prove non-membership with negligible probability.

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. SIM generates public parameters PK^* using the same algorithms as the real NSEC5 system. SIM generates (online) responses to queries for $x \in R$ that are identically distributed to those constructed by the real NSEC5 system during its setup phase. However, for every query $x \notin R$, SIM responds with a randomly-generated pair of values (y_j, y_{j+1}) (and its signature), rather than values (y_j, y_{j+1}) computed by applying equations (2),(3) to the elements of R, as in the real NSEC3 system. Fortunately, we argued in Lemma III.2, that a polynomial time adversary cannot distinguish between $\{F(x_i) | x_i \in R\}$ (per equations (2),(3)) and a collection of $|R|$ random values in

$\{0,1\}^n$, with more than a negligible advantage. It follows that the simulation cannot be distinguished from the real NSEC5 system. ■

Remark. 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 (*'no'*, $(S(x), S(x^*), x^*, v(x^*), Sig(x^*, v(x^*)))$). A resolver would verify this proof by checking that $F(x) = F(x^*)$ and verifying the signature $Sig(x^*, v(x^*))$. This would leak information about the set R (the fact that $x^* \in R$ and its value is $v(x^*)$), but this will not violate our privacy requirement since collisions occur with negligible probability.

IV. On-Line Public-Key Operations are Necessary

In this section we show that the secondary responders in a PSR system must perform a public-key computation *on each query*. We do so by showing how to obtain a public-key signature scheme from a PSR system, with the complexity of the signer is roughly equal to the complexity of the secondary nameserver (plus the complexity of the query algorithm). The signature scheme will be secure as long as the PSR system is complete, sound, and private—even if it satisfies only the weaker notion of privacy in Definition II.5 rather than the full-fledged zero-knowledge of Definition II.4.

Since in a public-key signature system, the signers must perform a public-key operation for each message, the same holds for the PSR system. Of course, a limited number of signatures can be precomputed in a signature scheme, and the same holds for a PSR system (*e.g.,* all positive responses may be precomputed, as in most existing constructions, including ours). The rest—and, in particular, negative responses to unexpected queries—must be done on-line.

We obtain signatures only when the PSR system satisfies the following property: the query algorithm is deterministic or, if randomized, soundness holds even when the adversary has the knowledge of the random values used to generate q. Note that the systems in which $q = x$ (such as our proposal and all the versions of DNS/DNSSEC) trivially satisfy this property.

Moreover, 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 a sender transmits an authentic 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 responder 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.

Both our transformations are in the random oracle model.

We first describe our result on transforming PSR schemes to PKA schemes, and then extend to signatures when the constraints on the query algorithm are satisfied.

A. Public-Key Authentication from PSR

Defining Public-key Authentication Security Public-key authentication (PKA; see [DDN00, Section 3.5]) can be seen as a relaxation of signature schemes in which we tolerate interaction between the sender and the receiver and 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 alive. (Such protocols can be used, for example, with key cards as provers in order to control access.) Identification protocols can be constructed from any zero-knowledge proof of knowledge [FFS87] for a computationally hard problem, but in practice no protocol where the efficiency of the prover is better than that of the signer in a signature scheme is known for either PKA or identification.

We define the relevant selective and existential security notions for public key authentication protocols.

Definition IV.1. ***Public key authentication security against selective forgery***. *A public key authentication 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 algorithm* A *playing against a challenger wins the game that will be described next with probability at most* ε.

1) *The forger* A *starts by picking a target message* M.
2) *The challenger runs the setup algorithm for the PKA, sends* PK *to the forger* A *and keeps* SK *secret.*
3) *Algorithm* A *mounts an adaptive chosen message attack by sending messages to be authenticated by the challenger,* $M_1, .., M_m$*, where* $\forall i : M_i \neq M$ *and for each one they engage in an authentication session.*
4) *At some point of* A*'s choosing it attempts to authenticate the message* M *to a verifier where* A *plays the role of the prover. Note that the sessions of authentication of the* M_i*'s may be running concurrently.*

We say that A *wins the game if the verifier accepts the authentication on* M.

Definition IV.2. ***Public key authentication security against existential forgery***. *A Public key authentication protocol* $(Setup, Prove, Verify)$ *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* A *can choose* M *at any point in the game.*

From PSR to Selectively Secure PKA We show how we can use a PSR system $(PSR\_Setup, PSR\_Query, PSR\_Answer, PSR\_Verify)$ that is selectively secure against polynomial time adversaries (as in Definition II.5) and construct a Public key authentication protocol

$$(PKA\_Setup, PKA\_Prove, PKA\_Verify)$$

that is selectively secure against polynomial time adversaries.

- $PKA\_Setup(1^k)$: Select uniformly at random a message $M_R \in U$, define $R = \{M_R\}$ and denote $v(\cdot)$ as the function that returns 1 on M_R and $\perp$ otherwise. Run the setup algorithm $PSR\_Setup(R, v(\cdot), 1^k)$ for the PSR, and obtain (PK, I_S), which will be our public and secret keys.

- $PKA\_Prove(M_i, I_S, PK)$: The prover acts as the *secondary* in the PSR system proving that $M_i \notin R$. It receives q, run the PSR_Answer algorithm, and sends back its results.
- $PKA\_Verify(M_i, PK)$: The verifier acts as the *resolver* in the PSR system. If runs the PSR_Query algorithm to obtain and send q, receives the response, and accepts if the PSR_Verify algorithm accepts the proof of non-membership.

Remark IV.3. Note that our PKA construction does not satisfy perfect completeness (if the verifier happens to choose M_R, we cannot authenticate that message). If the PSR system has perfect completeness, then we can also get a PKA system with perfect completeness by adding a bit to each element in the universe indicating whether it is 'real' or 'dummy', where we authenticate only the real elements. The set R should contain a single random dummy element and this way we can authenticate all real elements.

Theorem IV.4. *Suppose we have a PSR system that is ε-secure against* selective membership under an adaptive chosen message attack *then the derived Public key authentication protocol described above is ε'-secure against* selective forgery under an adaptive chosen message attack*, where $\varepsilon' = 4\varepsilon + \mu_s$ and μ_s is the soundness parameter of the PSR.*

Proof: Suppose that there exists a polynomial time forger B which manages to win the selective forgery security game for the derived PKA game with non-negligible probability ε'. We describe an adversary A that uses the forger B as a subroutine to win the selective membership security game against the PSR in polynomial time with a non-negligible advantage $\varepsilon = \frac{\varepsilon'}{4} - \frac{\mu_s(k)}{4}$.

- The adversary A starts by obtaining the message M which B selects to forge. A draws at random a message M^*, sets the target set to be empty, $S = \phi$, denotes $v(\cdot)$ as the function that returns $v(M) = v(M^*) = 1$ and $\perp$ otherwise and sends $(S, v(\cdot), M, M^*)$ to the challenger.
- The challenger defines $R = \{M\}$ with probability $\frac{1}{2}$ and $R = \{M^*\}$ otherwise. Next the challenger runs $PSR\_Setup(R, v(\cdot), 1^k)$ and sends PK to A.
- After algorithm A receives the public key PK from the challenger it emulates B by acting as an intermediary between the challenger and B by relaying their authentication messages to each other.
- Finally B plays the role of a prover and tries to forge an authentication for M, where A plays the role of the verifier. If the verifier A accepts the authentication, then A returns 1 (which means A believes $R = \{M^*\}$), else A chooses a bit uniformly at random and returns it.

If $R = \{M^*\}$, then B witnesses exactly the same view as in a real execution: the PSR_Setup algorithm is defined as in the PKA protocol as well as the remaining parts. In this case B wins his game with probability at least ε', and A identifies the success of the forgery attempt. So the probability A wins in this case is at least $\varepsilon' + \frac{1-\varepsilon'}{2}$, as either B succeeds in forging the authentication (probability ε') or A guesses the bit correctly (probability $\frac{1-\varepsilon'}{2}$). If $R = \{M\}$, then it is no longer true that B sees the same view as in a real execution, however, due to the PSR's *soundness* property the probability that B can generate a proof for a false statement (and $M \notin R$ is false in that case) is at most $\mu_s(k)$ (which should be negligible). So the probability A wins in this case is at least $\frac{1-\mu_s(k)}{2}$. Since these two cases are equally likely, this means that A wins the game with probability at least

$$\frac{1}{2}\left(\varepsilon' + \frac{1-\varepsilon'}{2}\right) + \frac{1}{2}\left(\frac{1-\mu_s(k)}{2}\right) = \frac{1}{2} + \frac{\varepsilon'}{4} - \frac{\mu_s(k)}{4}$$

which is a non-negligible advantage in winning the security game ($\varepsilon = \frac{\varepsilon'}{4} - \frac{\mu_s(k)}{4}$), in contradiction to the security assumption on the PSR system. ■

From Selective to Existential Security Next we prove that in the random oracle model using a PKA which is selectively secure (Definition IV.1) we can construct a PKA scheme which is existentially secure (Definition IV.2), thus showing that a PSR system implies a strong security notion for a PKA scheme. To do that we simply use a random oracle to hash the message we want to authenticate before authenticating it and modify the algorithms appropriately. As a result, the running time of the PKA sender will be greater than the running time of a PSR system secondary by only a single random oracle query.

Theorem IV.5. *Suppose that we have a Public key authentication protocol $(Setup, Prove, Verify)$ which is ε'-secure against* selective forgery under an adaptive chosen message attack *then in the random oracle model the derived scheme above 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 B which wins the existential security game for public key authentication in the random oracle model with non negligible probability ε. We use this adversary B to win the selective security game, contradicting our assumption.

Note that as we are in the random oracle model (see Appendix A) we control the random oracle and every time B wants to compute some $h(M)$ it gets the value. As adversary B runs in polynomial time, we know B can make at most a polynomial number of queries to the random oracle, assume an upper bound on that number is $q(k)$. We describe adversary A which uses adversary B in order to win the selective security game:

- Our adversary A chooses uniformly at random a message M from the message space and declares M as the message it intends to forge. A also draws at random $j \in [q(k)]$.
- The challenger simply runs the setup algorithm for the authentication protocol and sends A the public key PK.
- A starts by emulating B, by functioning as an intermediary between the challenger and B and relaying their authentication messages to each other. When B queries the random oracle h, answer with random values at all steps except the j^{th} one. At the j^{th} step, when B queries h for message m', then set $h(m')$ to be M. If

at some point before the forgery attempt by B, it asks to authenticate the message m', A stops and declares failure.
- Finally B tries to forge an authentication for some message M^*, if $h(M^*) = M$ then A uses this forged authentication to try and authenticate M, else it fails.

We may assume that B accesses h on the message it tries to forge (otherwise its probability of success is negligible). Therefore with probability $\frac{1}{q(k)}$ adversary A sets the value of the random oracle over the message B tries to forge; $h(M^*)$, to be the message M that A tries to forge as well. This means that A wins the security game in the case that both B managed to successfully forge an authentication for his target message and the right j was picked, which happens with probability $\varepsilon' = \frac{\epsilon}{q(k)}$. ■

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 complexity. Our goal in this section is to point out that for PSR systems that satisfy some restrictions on the query algorithm, we can actually get a signature scheme of about the same complexity, as well. This shows that on-line public-key operations on the responders are inherent for those schemes, including any scheme that simply sends x as the query.

Consider any PSR system in which the Query algorithm is deterministic and apply the transformation of Section IV-A to get a PKA scheme. Observe that interaction is not necessary in the resulting PKA scheme, because each side can compute the receiver'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.

In case the Query algorithm is randomized, the same approach does not work, because the PKA sender cannot be trusted to choose or even to know the randomness that the Query algorithm uses. We get around the problem of choosing the randomness by using the approach of Fiat and Shamir [FS86]: 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 sender to know the randomness and thus to compute the receiver'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.6. *Consider any PSR system* $(PSR\_Setup, PSR\_Query, PSR\_Answer, PSR\_Verify)$ *for which the* PSR_Query *algorithm is deterministic or for which the randomness of the* PSR_Query *can be given to the adversary without harming soundness. Such a system implies an existentially unforgeable digital signature scheme whose signing complexity is equal to the complexity of* PSR_Query *plus* PSR_Answer *(plus at most two random oracle queries).*

Remark IV.7. *Note that although we separated the nameservers into 2 parties (a primary and secondaries), we still would have gotten a signature scheme with a 2 party protocol* i.e., *when we have just a primary that commits to R and $v(\cdot)$ and* ***honest*** *secondaries, or that the primary itself is in charge of providing proofs for resolvers. This means that even without the separation of nameservers the prover still has to generate signatures online in order to prevent zone enumeration attacks, while providing soundness.*

C. Discussion

We have shown that we can use a PSR system satisfying the zero-knowledge requirement (Definition II.4) and hence the selective membership requirement (Definition II.5) in order to build signatures, PKA and identification schemes. We therefore want to claim that we demonstrated that the work involved in this task must be non-trivial, unlike the NSEC3 protocol which only uses hashing but does not prevent zone enumeration. One could protest and argue that our zero-knowledge requirement or even the selective membership requirement are too strong and it may be possible to have a more relaxed notion of privacy that still prevents zone enumeration. We now argue that this is not the case.

Suppose we modify the privacy notion and protect against an adversary that produces an element it did not explicitly query on (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 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 he believes to be in the set R which he has not explicitly queried (this is 'success'), there is a simulator that interacts with an oracle to the set R, which is successful as well with similar probability, where similar means that the difference is negligible. We can show that under this requirement we get 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, under a similar reduction to 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 we generalize the constructions of this paper and show how to obtain PSR systems without random oracles. We suggest a general construction based on VRFs [MRV99] and in particular relatively efficient incarnations of it [DY05], [HW10]. We also provide a construction based on *hierarchical identity based encryption* and in particular the one by Boneh, Boyen and Goh [BBG05] which does not reveal any information about the set R, even not its cardinality. For both constructions the amount of work consists of a few bilinear operations and logarithmic in $|U|$ number of multiplications.

We also plan to write an Internet Draft for NSEC5.

VI. Acknowledgements

We thank Casey Deccio, Daniel Kahn Gillmor, Ben Laurie, Jared Mauch, Matthijs Mekking, Benno Overeinder and Wouter Wijngaards for useful discussions.

References

[AA04] D. Atkins and R. Austein, *Threat Analysis of the Domain Name System (DNS)*, RFC 3833, Internet Engineering Task Force, August 2004.

[AAL+05a] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *DNS Security Introduction and Requirements*, RFC 4033, Internet Engineering Task Force, March 2005.

[AAL+05b] ———, *Protocol Modifications for the DNS Security Extensions*, RFC 4035, Internet Engineering Task Force, March 2005.

[AAL+05c] ———, *Resource Records for the DNS Security Extensions*, RFC 4034, Internet Engineering Task Force, March 2005.

[Ait11] Brian Aitken, *Interconnect communication MC / 080:DNSSEC Deployment Study*, http://stakeholders.ofcom.org.uk/binaries/internet/domain-name-security.pdf, 2011.

[AKB07] R. Arends, M. Kosters, and D. Blacka, *DNS Security (DNSSEC) Opt-In*, RFC 4956, Internet Engineering Task Force, July 2007.

[AL01] Paul Albitz and Cricket Liu, *Dns and bind*, O'Reilly Media, Inc., 2001.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh, *Hierarchical identity based encryption with constant size ciphertext*, in Cramer [Cra05], pp. 440–456.

[BEG+94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor, *Checking the correctness of memories*, Algorithmica **12** (1994), no. 2/3, 225–244.

[Ber11] Daniel J. Bernstein, *Nsec3 walker*, http://dnscurve.org/nsec3walker.html, 2011.

[BM10] Jason Bau and John C. Mitchell, *A security evaluation of dnssec with nsec3*, NDSS, The Internet Society, 2010.

[BR93] Mihir Bellare and Phillip Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM Conference on Computer and Communications Security, ACM, 1993, pp. 62–73.

[BR94] ———, *Optimal asymmetric encryption*, EUROCRYPT, Lecture Notes in Computer Science, vol. 950, Springer, 1994, pp. 92–111.

[BR96] ———, *The exact security of digital signatures — how to sign with RSA and Rabin*, EUROCRYPT, Lecture Notes in Computer Science, vol. 1070, Springer, 1996, pp. 399–416.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi, *The random oracle methodology, revisited*, J. ACM **51** (2004), no. 4, 557–594.

[CHL+05] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin, *Mercurial commitments with applications to zero-knowledge sets*, in Cramer [Cra05], pp. 422–439.

[Cor00] Jean-Sébastien Coron, *On the exact security of full domain hash*, CRYPTO, Lecture Notes in Computer Science, vol. 1880, Springer, 2000, pp. 229–235.

[Cra05] Ronald Cramer (ed.), *Advances in cryptology - eurocrypt 2005, 24th annual international conference on the theory and applications of cryptographic techniques, aarhus, denmark, may 22-26, 2005, proceedings*, Lecture Notes in Computer Science, vol. 3494, Springer, 2005.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor, *Nonmalleable cryptography*, SIAM J. Comput. **30** (2000), no. 2, 391–437.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy, *A verifiable random function with short proofs and keys*, Public Key Cryptography, Lecture Notes in Computer Science, vol. 3386, Springer, 2005, pp. 416–431.

[FFS87] Uriel Feige, Amos Fiat, and Adi Shamir, *Zero knowledge proofs of identity*, STOC, ACM, 1987, pp. 210–217.

[FS86] Amos Fiat and Adi Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, CRYPTO, Lecture Notes in Computer Science, vol. 263, Springer, 1986, pp. 186–194.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali, *How to construct random functions*, J. ACM **33** (1986), no. 4, 792–807.

[GM12] 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.

[GM14] ———, *Authenticated Denial of Existence in the DNS*, RFC 7129, Internet Engineering Task Force, February 2014.

[Gol01] Oded Goldreich, *The foundations of cryptography - volume 1, basic techniques*, Cambridge University Press, 2001.

[Gol04] ———, *The foundations of cryptography - volume 2, basic applications*, Cambridge University Press, 2004.

[HW10] Susan Hohenberger and Brent Waters, *Constructing verifiable random functions with large input spaces*, EUROCRYPT, Lecture Notes in Computer Science, vol. 6110, Springer, 2010, pp. 656–672.

[Kam11] Dan Kaminsky, *Phreebird*, http://dankaminsky.com/phreebird/, 2011.

[KMG12] O. Kolkman, W. Mekking, and R. Gieben, *DNSSEC Operational Practices, Version 2*, RFC 6781, Internet Engineering Task Force, December 2012.

[LSAB08] B. Laurie, G. Sisson, R. Arends, and D. Blacka, *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*, RFC 5155, Internet Engineering Task Force, March 2008.

[MHKS14] Andrew Miller, Michael Hicks, Jonathan Katz, and Elaine Shi, *Authenticated data structures, generically*, POPL, ACM, 2014, pp. 411–424.

[MRK03] Silvio Micali, Michael O. Rabin, and Joe Kilian, *Zero-knowledge sets*, FOCS, IEEE Computer Society, 2003, pp. 80–91.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan, *Verifiable random functions*, FOCS, IEEE Computer Society, 1999, pp. 120–130.

[Nie02] Jesper Buus Nielsen, *Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case*, CRYPTO, Lecture Notes in Computer Science, vol. 2442, Springer, 2002, pp. 111–126.

[NN00] Moni Naor and Kobbi Nissim, *Certificate revocation and certificate update*, IEEE Journal on Selected Areas in Communications **18** (2000), no. 4, 561–570.

[Pow13] PowerDNS, *Powerdns manual*, December 2013.

[RS05] Venugopalan Ramasubramanian and Emin Gün Sirer, *Perils of transitive trust in the domain name system*, Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, USENIX Association, 2005, pp. 35–35.

[San04] Marcos Sanz, *Dnssec and the zone enumeration*, European Internet Forum: http://www.denic.de/fileadmin/public/events/DNSSEC_testbed/zone-enumeration.pdf, October 2004.

[Sho01] Victor Shoup, *OAEP Reconsidered*, CRYPTO, Lecture Notes in Computer Science, vol. 2139, Springer, 2001, pp. 239–259.

[SL06] G. Sisson and B. Laurie, *Derivation of DNS Name Predecessor and Successor*, RFC 4471, Internet Engineering Task Force, September 2006.

[TT10] Roberto Tamassia and Nikos Triandopoulos, *Certification and*

authentication of data structures, AMW, CEUR Workshop Proceedings, vol. 619, CEUR-WS.org, 2010.

[WI06] S. Weiler and J. Ihren, *Minimally Covering NSEC Records and DNSSEC On-line Signing*, RFC 4470, Internet Engineering Task Force, April 2006.

[WSBW14] Matthaus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis, *Gpu-based nsec3 hash breaking*, 2014 IEEE 13th International Symposium onNetwork Computing and Applications (NCA), IEEE, 2014, pp. 137–144.

APPENDIX

A. The Random Oracle model

As our construction is analyzed in the random oracle model we need to rigorously define this model. The random oracle model has been used quite extensively to analyze cryptographic protocols [BR93], [BR94], [BR96], [Cor00], [Sho01]. We define the model as in Canetti, Goldreich and Halevi [CGH04]. In a scheme set in the Random Oracle Model, all parties including adversaries interact with each other like they would at the standard model, but they can also make oracle queries. According to the security parameter k and a length function $\ell_{out}(\cdot)$, an oracle O is a function chosen uniformly at random out of all possible functions mapping $\{0,1\}^*$ to $\{0,1\}^{\ell_{out}(k)}$. Every party has access to this oracle. Security is defined as usual, meaning that a system is still considered secure when its adversary has a negligible probability of success or a negligible advantage, where the probability is also taken over the choices of the random oracle. Note that in the proof of security the random oracles can be "programmed", meaning that certain values of the random oracle can be set either before hand or on the fly to be specific values (chosen uniformly at random) by a simulator (see Nielsen [Nie02]). Values can be set only the first time someone wishes to know $O(x)$ as the oracle must remain consistent.

B. Signature schemes

We use signature schemes in our construction, for that end we define signature schemes and their properties as we need them for our constructions. We define public key signature schemes as in Goldreich [Gol04].

Definition A.1. *A signature scheme is defined by three (polynomial time) algorithms (G, S, V): The key generator G gets the security parameter k and outputs two keys, a signing key sk and a verification key vk, $G(1^k) = (sk, vk)$. 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 vk 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, vk) 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*

$$Pr[V_{vk}(S_{sk}(M), M) = 1] = 1$$

We will assume that the signature scheme is deterministic in the sense that for every message m there is a single signature σ that the signing algorithm produces (even though the verification algorithm may accept many different signatures). This is true wlog because we can always add to the signing key sk a description of a pseudorandom function to provide the randomness needed to sign m (see [GGM86]).

The type of security we require from our signature scheme is "existential unforgeability against chosen message attacks", which means that even an adversary who can gain access to a polynomial number of signatures to messages of his choosing will still not be able to generate a signature for any message the adversary did not explicitly request a signature for.

Definition A.2. *A signature scheme is existentially secure against chosen message attacks if every probabilistic polynomial time adversary A wins the following security game with negligible probability. The game is modeled as a communication game between the adversary and a challenger C.*

- *The challenger C runs the setup algorithm $S(1^k)$ and obtains (sk, vk), sends vk 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 in return 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 $V_{vk}(s, M) = 1$, i.e., *the forged signature is accepted as valid.*

C. RSA and Trapdoor Permutations

Our construction needs a trapdoor permutation and we use the famed RSA function. An RSA scheme has three algorithms (G, RSA, RSA^{-1}). The key generator G gets the security parameter k and outputs two keys, a public key PK (used for the forward direction: encryption and verifying signatures) and a secret or private key SK (used for the backward direction: decryption and signing). The algorithm G chooses an exponent e (for efficiency we could select e to be small, say 3), two large prime numbers P and Q of length roughly k such that e is relatively prime to $P-1$ and to $Q-1$ and computes $N = P \cdot Q$. It then calculates d such that for $L = lcm(P-1, Q-1)$ it holds that $d \cdot e \equiv 1 \bmod L$. It then sets $PK = (N, e)$ and $SK = (N, d)$. The RSA forward algorithm takes a value $m \in \mathbb{Z}_N$ and the public key and computes $RSA_{PK_{rsa}}(m) \equiv m^e \bmod N \equiv \sigma \bmod N$. The RSA backward algorithm takes a value $\sigma \in \mathbb{Z}_N$ and the secret key and computes $RSA^{-1}_{SK_{rsa}}(\sigma) \equiv \sigma^d \bmod N \equiv m \bmod N$.

Here are a few known properties/assumptions of this encryption scheme which we will find useful.

RSA is a permutation. Every value $x \in \mathbb{Z}_N$ is mapped by the encryption algorithm to some unique $y \in \mathbb{Z}_N$ and the decryption algorithm maps y back to x. Ideally we would like the domain and range of the RSA to be $\mathbb{Z}^*_N$ in order not to expose any integers which are not relatively prime to N as they would expose the factorization of N and by that the secret

key of the RSA. If we have to "artificially" remove elements from the domain $\mathbb{Z}_N$, due to the fact they are not relatively prime to N that will also reveal those dangerous integers. Thus we choose $\mathbb{Z}_N$ as the domain and range knowing that those integers are rare; $\frac{P+Q}{P \cdot Q} = \frac{1}{P} + \frac{1}{Q}$ fraction of the domain $\mathbb{Z}_N$.

The RSA hardness assumption wrt to exponent e and security parameter k. The assumption states that it is hard to compute the RSA inverse of a random value: for any polynomial time adversary A, for exponent e, random primes P, Q of length k where e is relatively prime to $P-1$ and $Q-1$ and $N = P \cdot Q$, for a random $y \in \mathbb{Z}_N$, it holds that

$$\Pr[A(y, N, e) = x \; and \; x^e \equiv y \bmod N]$$

is negligible in the security parameter.

Note that succeeding in finding the RSA inverse of any element of a set of r random challenges is just as hard. The reason is that given a single random z, by selecting random $w_i \in \mathbb{Z}_N$ and generating $z_i = z \cdot w_i^e \bmod N$ we get a set of r numbers so that from the RSA inverse of any of them it is possible to get $RSA^{-1}(z)$.

RSA is efficient. We can use low exponent RSA encryption in our construction in order to increase efficiency. If we pick e to be small then the forward algorithm will work fast, as it will need to make a smaller number of modular multiplications. The inversion algorithm takes the same amount of time regardless of the size of e.