NSEC5: Provably Preventing DNSSEC Zone Enumeration

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

***Abstract*—This paper uses cryptographic techniques to study the problem of 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*, where an adversary launches a small number of online DNSSEC queries and then uses offline dictionary attacks to learn which domain names are present or absent in a DNS zone. We explain why the current DNSSEC standard (with NSEC and NSEC3) suffers from zone enumeration: we use cryptographic lower bounds to prove that DNSSEC's three design goals — high performance, security against network attackers, and privacy against zone enumeration — cannot be satisfied simultaneously. We then introduce NSEC5, a new cryptographic construction that solves the problem of DNSSEC zone enumeration while matching our lower bounds and remaining faithful to the operational realities of DNSSEC. NSEC5 can be thought of as a variant of NSEC3, where the hash function is replaced with an RSA-based keyed-hashing scheme.**

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, RFC 5155) 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 strongly implies 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: {`golbe,dipapado,reyzin,sachinv`}`@cs.bu.edu`.

†Weizmann Institute of Science, Department of Computer Science and Applied Mathematics. Email:{`moni.naor,asaf.ziv`}`@weizmann.ac.il`.

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, from query: "What is the IP address for `www.example.com`?" to 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. (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. The trivial idea of precomputing signed responses of the form "______ is a non-existent domain" also fails, since the number of possible queries that deserves such a response is infinite, making precomputation of signed responses infeasible. Instead, RFC4033 defined NSEC, a precomputed denial-of-existence record technique, 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

[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].

of existent names lexicographically before and after the non-existent name (`w.example.com` and `z.example.com`), as well as its associated DNSSEC signatures. 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 (see RFC 4033), before eventually arriving at consensus that it is a problem from some zones (see RFC 5515). 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 5515:

> 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 suggested NSEC3, a precomputed denial of existence technique, designed to make zone enumeration more difficult. With NSEC3, all domain names present in a zone are first hashed, and then their hashes 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 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.[2]

Hashing the names (with a one-way function) makes trivial enumeration of the zone much more difficult. However, the savvy reader might have noticed that the NSEC3 design is still 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. In fact, 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.

[2]Following this, there was also an Internet Draft [GM12] (that has since 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.

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, NSEC3 requires salt to be *common to the entire zone*. Since changing the salt requires re-computing the signatures for the entire zone, RFC 6781 recommends updating the salt only when key-rollover takes place (a very 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. Indeed, RFC 5155 acknowledges this: "The NSEC3 RRs are still susceptible to dictionary attacks (i.e., the attacker retrieves all the NSEC3 RRs, then calculates the hashes of all likely domain names, comparing against the hashes found in the NSEC3 RRs, and thus enumerating the zone)."

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

Our first contribution is the following cryptographic model:

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

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.

(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, as was laid out in RFC 5155 and the quote above.

(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.

The formal cryptographic model and security definitions are in Section II. We define Primary-Secondary-Resolvers (PSR) systems for proving membership and non-membership in a set.

D. Cryptographic Lower Bound

The DNSSEC standard (with NSEC or NSEC3) has resolvers send queries for names in the clear, and limits the computation of secondary nameservers to a few cryptographic hashes. We demonstrate in Section IV that if the resolvers send queries in the clear, then satisfying both the soundness and privacy goals implies that nameservers must *necessarily* compute a public-key cryptographic signature for each negative response. Moreover, even if the resolvers pre-process the query (rather than send it in the clear), 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-E we discuss the question of whether our privacy requirements are "too strong" and argue that any system that prevents zone enumeration 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.

E. NSEC5: A Denial-of-existence Mechanism

Armed with the knowledge that privacy associates an online signature computation with 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 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. 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, PK_S corresponding to SK and SK_S will, naturally, be provided to the resolver, using the standard mechanisms used to transmit public keys in DNSSEC.

We emphasize that privacy makes sense only when the secondary nameserver can keep some information secret (else, R is no longer private). Thus, the addition of SK_S to the secondary nameserver does not introduce any additional security vulnerability: if it is leaked, soundness is not compromised.

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.

For each record x that is present in the zone R, the primary resolver computes

$$S(x) = RSA^{-1}_{SK_S}(h_1(x)) \qquad F(x) = h_2(S(x))$$

where h_1, h_2 are hash functions and $RSA^{-1}_{SK_S}$ is the RSA signature (or decryption), keyed with the *secondary* key SK_S. The resulting F values are lexicographically ordered, and each pair is signed by the *primary nameserver* using its key SK (just like in NSEC and NSEC3). The resulting pair of F values is an NSEC5 record.

To prove the non-existence of 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) 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 first using PK_S to (1) check that $S(q)$ is a valid RSA signature on $h_1(q)$, (2) confirm that the NSEC5 record is validly signed by PK, and (3) check that $h_2(S(q))$ is lexicographically between the hashes in the NSEC5 record. In other words, $S(q)$ maintains soundness by acting as a "proof" that the value $F(q)$ is the correct "keyed hash" of q.

Security and Privacy. In Section III we formally describe the NSEC5 scheme and prove that our construction satisfies both soundness and privacy as defined in Section II. Privacy follows because the resolver does not know the secondary 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!). Meanwhile, integrity follows 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 computed $S(q)$ correctly, and responded with the right NSEC5 record.

Performance. Our solution, NSEC5, 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.

Finally, we note that online signing for denial of existence was already proposed in RFC 4470 (and further discussed in RFC 4471). RFC 4470 suggested that every nameserver (even the secondary) be given the primary key for the zone, and used it to produce online signatures to responses of the form "q is a non-existent domain". While some dismissed the RFC 4470 solution because it compromised soundness, our solution has the same computational complexity and requires no compromise of soundness. Indeed, online signing in DNSSEC is already a very real possibility; for example, the powerDNS nameserver supports online DNSSEC signing [Pow13, Sec. 4] and the same is true for Dan Kaminsky's Phreebird DNS proxy [Kam11]. We therefore believe it presents an attractive alternative to NSEC3 for those zone operators that 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-in or the space-saving techniques offered by 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. 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. We can't use the existing ZKS solutions as is, however, because even the best known constructions of ZKS [CHL$^+$05] are too inefficient to be practical for DNSSEC[3]. On the other hand, the requirements in a ZKS are very stringent, in that one does not trust even the *primary resolver* (*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]). 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, or PSR for short, with the goal of getting denial of existence with zone enumeration prevention. A PSR is an interactive proof system consisting of three parties. The *primary*, sets up the system by committing to a "privileged" set $R \subseteq U$ (existent domain names in the zone) where U is the universe of elements (domain names) and a corresponding value $v(x) \in V$ for every $x \in R$ (*e.g.,* IP address). It then publishes a public key for the system PK, which should be known to both the *secondaries* and the *resolvers* in the system (via the usual DNSSEC mechanisms). The *secondaries* also get some parameters I_S. We divide I_S into two parts; the first is DS standing for data structures, allowing quick search and certifications, which we can tolerate if it leaks to *resolvers*, as this is information that will naturally leak during execution of the protocol. The second part is SS for which it is critical that it remains secret, *e.g.,* cryptographic keys. If a *resolver* happens to learn SS it could enumerate over the set R independently without the help of *secondaries*, but if the *resolver* only learns DS it will not help him get any additional power in figuring out the set R. After the setup phase is complete, the *secondaries* and *resolvers* get their keys and parameters and function as a prover and verifier of statements of the sort "$x \in R$ and $v(x) = y$" or "$x \notin R$".

Following the design of DNSSEC, we only consider two round protocols: where a query is sent from the *resolver* to the *secondary* and a response is returned. More interactive protocols are possible, but we do not consider them here. DNSSEC standard has *resolvers* sending queries in the clear, thus we avoid the case where a *resolver* uses secret coins in his queries, meaning the case where a *resolver* issues a query for $x \in U$ using some randomness s_q which it stores and later uses to verify the response it gets from the *secondary* is correct. We can still consider randomized queries, but not ones which use the randomness for the verification procedure as well, *i.e.,* public coins.

A. PSR Systems

The system consists of four algorithms:
The *Setup* algorithm is used by the *primary* to generate the public key PK, which it publishes to all parties in the protocol and the information $I_S = (DS, SS)$, delivered to the *secondaries*. The *resolvers* use the *Query* algorithm to generate queries for elements in the universe which they send to a *secondary*, who replies to the queries 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 $v : R \to V$[4], this algorithm outputs two strings: PK, a public key and $I_S = (DS, SS)$ the parameters given to the *secondaries*.

[3] [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".

[4] This function will be used to map domain names to their corresponding IP addresses.

I_S contains two parts; the secret information SS and DS the data structure information. It outputs (PK, I_S).

$Query(x, PK)$

On input $x \in U$ and the public key PK this algorithm outputs a query q from which one can deduce efficiently the element x. The query generation may either be deterministic or 'public-coins' where the algorithm is randomized but does not output any secret information about the query.

$Answer(q, I_S, PK)$

The algorithm gets as input a query q for some element $x \in U$, the public key and parameters. If $x \in R$ then the algorithm outputs a bit $b = 1$ and a proof π for $x \in R$ and $v(x)$, else it outputs $b = 0$ and a proof π for $x \notin R$.

$Verify(x, q, b, \pi, PK)$

The algorithm gets as input $x \in U$, a query q for this element x, a bit b, a proof π and the public key PK. If $b = 1$ then it checks that the proof π validates that $x \in R$ and the value is $v(x)$ and if $b = 0$ it checks to validate that $x \notin R$. If the proof is correct it returns 1 and otherwise 0.

For simplicity, when we defined the system above we only consider the case where the set R is static. It is determined when the *primary* sets up the system and we cannot change it afterwards. There are methods for handling it that borrow from the CRL world (*e.g.,* we could use the Naor-Nissim certificate update and revocation scheme [NN00]) but we chose not to concentrate on this aspect in this work.

We will require the above four algorithms to satisfy three properties.

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 $\forall x \in U$,*

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

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

As for security, or *soundness*, we want that even a malicious *secondary* in the system would not be able to convince an honest *resolver* of a false statement with more than a negligible probability.

Definition II.3. ***Soundness:*** *for all probabilistic polynomial time adversaries A and for all $x \in R$ we have*

$$\Pr\left[\begin{array}{l}(PK, I_S) \overset{R}{\leftarrow} Setup(R, v(\cdot), 1^k);\\ (q, \pi) \overset{R}{\leftarrow} A(PK, I_S) :\\ Verify(x, q, 0, \pi, PK)\end{array}\right] \leq \mu(k)$$

and for all $x \notin R$ we have

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

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

Even though we don't require perfect completeness or soundness, our NSEC5 system satisfies more stringent requirements; an adversary cannot find x violating either completeness or soundness even after getting the public-key PK and I_S.

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 a PSR system to have a simulator with oracle access to the set R which receives $f(R)$ and can fool a *resolver* into believing it is communicating with a real system. Later, in Section II-D we show that the f-zk notion implies a more "intuitive" security definition.

The idea behind our f-zk notion is that a *resolver* learns nothing from the responses it gets from the *secondaries*, besides the response to his query and the information $f(R)$, which might leak during the protocol's execution. We require that the *resolver* cannot distinguish between: (1) a real system which provides the original proofs, and (2) a simulator that can only obtain the answer to each *resolver*'s query, but must still be able to "forge" a satisfactory proof for that response. The use of such a simulator allows us to deduce the *resolver* has not learned much about R from the proofs; if he had, he would be able to distinguish between an interaction with the simulator and one with the real *secondary* (at least after he gets R explicitly). The use of simulators in order to prove that a protocol is zero knowledge is standard in cryptography (see [Gol01] Chapter 4 for a more comprehensive treatment of ZK and simulators).

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 ask on point x only when the adversary explicitly queries on an element x. On its first step SIM receives $f(R)$ and outputs a fake public key PK^*, a fake secret key SK_{SIM} and the leaked information$f(R)$. On the following steps SIM receives queries from the adversary and needs to output a (simulated) proof of either $x \in R$ plus $v(x)$ or $x \notin R$; to do this, SIM is allowed to query the R-oracle for the element x. The simulator's output (public-key and proofs) should be computationally indistinguishable from the output generated by a real PSR system.

We divide this process into two phases. The first is an interactive protocol where the adversary communicates with the simulator or a PSR system. First the adversary gets the public key, either from a PSR system setup algorithm or from a simulator which gets $f(R)$. Then the adversary starts issuing queries q_i (adaptively), based on the public key and previous responses to queries it got. The simulator/PSR system responds to the queries with the answers (b_i, π_i) which the adversary can verify.

The second phase starts after the interactive protocol ends, where a distinguisher tries to tell apart the two views generated by the protocols. We say that the system is f-zk if for every adversary there exists a simulator such that no distinguisher who knows R can distinguish with more than a negligible advantage between the two views containing the public key, $f(R)$, queries and responses which were generated by either the system or the simulator.

The first step of the interactive protocol consists of the generation of keys, either by a PSR system:

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

or by the simulator that generates fake keys :

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

the rest is the interactive protocol of queries and responses described above, where the simulator uses the fake public key PK^* and the fake secret key SK_{SIM} to answer queries and the system uses the real keys (PK, I_S).

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

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

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

and

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

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

Remark. Note that the requirement of the simulation is online: there is no rewinding and the number of queries and nature of the queries to the R-oracle are restricted by the calls the *resolver* makes. This means our requirements for the simulator are weak, in the sense that the simulator only has the oracle access and $f(R)$ to work with, but still manages to provide indistinguishable proofs with overwhelming probability. The more power the simulator has, the easier it is to construct a valid simulator that can fool an adversary, making the f-zk property easier to achieve, thus using this basic definition for a simulator makes our ZK requirement stronger.

Also note that this concept of a simulator receiving some $f(R)$ may look similar to the definition of auxiliary input zero knowledge (see [Gol01] Chapter 4), but they are different. In the latter both the adversary and the simulator receive the same auxiliary information and we wish that the adversary is still unable to 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 this definition as well in case we would like our *resolvers* to have some prior information about the set R, but they would still not be able to gain any additional information on R besides $f(R)$ and the prior knowledge they received.

Note that the adversary is given the value $f(R)$ on its first step. Also note that we choose to define f-zk for a 2-round interactive PSR protocol. One can easily generalize the f-zk property to include more rounds.

D. Zero-knowledge Implies Hardness of Finding an Additional Element

We want to make sure that the zero-knowledge with respect to the *resolvers* implies that they indeed cannot obtain information about additional elements other than those that the *resolver* has explicitly queried. Like the case of zone enumeration, we wouldn't like an adversary to be able to enumerate over the zone without prior knowledge. Hence we consider an attack where the adversarial *resolver* tries to determine which of two a-priori known elements is in R, without querying for those two elements. We prove that the f-zk for $f(R) = |R|$ implies this attack can succeed only with negligible advantage. We call this *selective membership security*; it is defined by a game where an adversary needs to guess correctly a bit with non-negligible advantage in order to win.

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 $x_1, x_2 \notin S$ 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* no *$y_i \in \{x_0, x_1\}$. The challenger responds with proper answers to all the queries: $A_1, .., A_q$.*

4) *Finally A outputs one bit b, $b = 0$ if A believes that $x_0 \in R$ and $b = 1$ if it believes $x_1 \in R$.*

We say that A won the game if it guessed the bit b correctly.

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 as 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 $(S, v : R \to V, x_1, x_2)$ 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 $x_b \in R$, if it can find it with a non-negligible advantage then the adversary could also distinguish between the two views.

For $j \in \{1, 2\}$ 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 $\mu_f(k)$, 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 as the simulator SIM doesn't know the set R, it knows just its cardinality and it has an oracle to the set R which it uses whenever it is queried on some element and is not allowed to query any other elements. Thus during the key generation part both views are identically distributed as in both cases SIM gets the same $f(R)$ and cannot query its oracle. Note that the adversary is not allowed to query for x_0, x_1 because it is his target challenge, so the adversary can issue the same set of queries to the simulator and get the same answers to all of them. Thus both views are identically distributed and cannot be distinguished.

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

S_{rsa}, RSA, RSA^{-1}	RSA algorithms
PK_{rsa}, SK_{rsa}	RSA keys
S_{sig}, Sig, Ver	Signature scheme algorithms
PK_{sig}, SK_{sig}	Signature scheme keys
h_1	Random oracle from U to $[N]$
h_2	Random oracle from $[N]$ to $\{0,1\}^n$
$F : U \to \{0,1\}$	The function $h_2(RSA^{-1}_{SK_{rsa}}(h_1()))$
$S : U \to [N]$	The function $RSA^{-1}_{SK_{rsa}}(h_1())$
$R = \{x_1, .., x_r\}$	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.

III. NSEC5 Construction and Proof

We show a construction of an efficient PSR system based on RSA and a signature scheme and proved secure in the random oracle model. Table 1 summarizes our notation.

The three algorithms (S_{rsa}, RSA, RSA^{-1}) the setup, forward (encrypting or verifying) and backward (decrypting or signing) computation of the RSA trapdoor permutation for modulo N(see Appendix C). The corresponding keys PK_{rsa}, SK_{rsa} are the secondary keys of the scheme. Likewise, the three algorithms (S_{sig}, Sig, Ver) are the setup, signature and verification algorithms of an existentially unforgeable signature scheme (see Appendix B) and PK_{sig}, SK_{sig} are the corresponding keys, which are the primary keys of the scheme. We can consider the BLS signature scheme [BLS04] as it has very short signatures which are proven existentially secure and are efficient in the random oracle model (see Appendix B for the description of signature schemes). Functions h_1, h_2 are modeled as random oracles (See Appendix A). For h_2, the value of n is chosen to prevent birthday attacks. Finally, function F will in practice, look random for any observer not knowing the secret RSA key, and function S will be used to show that $F(x)$ was computed correctly.

As explained in the introduction, we compute F over the entire set R, sort the values lexicographically and sign every adjacent pair of values (y_j, y_{j+1}) with signature $Sign(y_j, y_{j+1})$. These $\{y_j\}_{j=0}^r$ and $\{Sign(y_j, y_{j+1})\}_{j=0}^r$ are given to the *secondary*. In order to respond to negative queries $x \notin R$, the *secondary* computes $F(x)$ and its proof $S(x)$ and sends $S(x)$ together with the pair (y_j, y_{j+1}) and the signature $Sign(y_j, y_{j+1})$ such that $F(x)$ is between y_j and y_{j+1}. The *resolver* can compute $F(x)$ by applying h_2 on $S(x)$, thus validating the response. Responses to positive queries are as in DNSSEC: we prove $x \in R$ by sending $(x, v(x))$ signed by the *primary*.

The four algorithms for the PSR system are:

Setup: the setup algorithm $Setup(R, v(\cdot), 1^k)$ gets the set R and the values v associated with it as well as a security parameter. It uses the Setup algorithm $S_{rsa}(1^k)$ to obtain

(PK_{rsa}, SK_{rsa}) for the RSA scheme and uses the setup algorithm for the signature scheme $S_{sig}(1^k)$ in order to obtain (PK_{sig}, SK_{sig}). Choose the two random oracles h_1 and h_2 as specified before where n is chosen to be large enough such that $\frac{|R|}{2^n}$ is negligible. The public key is defined to be $PK = (PK_{rsa}, PK_{sig}, h_1, h_2)$.

Now for every $x_j \in R$ calculate

$$y_j = F(x_j) = h_2(RSA^{-1}_{SK_{rsa}}(h_1(x_j)))$$

For convenience of notation we assume wlog that the x_j's are ordered lexicographically by the values of F over them, thus $y_1, .., y_r$ are lexicographically ordered and then we add $y_0 = 0^n$ and $y_{r+1} = 1^n$. Now for each $j \in \{0, .., r\}$ use the signature scheme to create a signature:

$$Sign(y_j, y_{j+1}) = Sig_{SK_{sig}}(y_j, y_{j+1})$$

Use the same signature scheme to compute for every $x_j \in R$:

$$Sign(x_j, v(x_j)) = Sig_{SK_{sig}}(x_j, v(x_j))$$

Define $DS = (\{Sign(y_j, y_{j+1})\}_{j=0}^r, \{y_j\}_{j=1}^r)$ and the secret information $SS = (SK_{rsa}, \{Sign(x_j, v(x_j))\}_{j=1}^r)$. The parameters given to the *secondaries* are $I_S = (DS, SS)$.

Query generation: algorithm $Query(x, PK)$ is a simple as it can be: output $q = x$ as a query for element $x \in U$. Note that the query doesn't contain any secret or random information.

Answering a query: $Answer(q, I_S, PK)$ is performed by first calculating $(F(q), S(q)) = (y, \pi_y)$ (recall that $q = x$). As the values $\{y_j\}_{j=1}^r$ are ordered lexicographically we check if there exists an index j for which $y = y_j$.

If we find an index j for which $y = y_j$, then we know $x \in R$ and find the signature $Sign(x_j, v(x_j))$ [5] which opens to $(x, v(x))$ and we return $('yes', Sign(x_j, v(x_j)), (x_j, v(x_j)))$.

If we can't find such a match that means that $x \notin R$ so we find the index j for which $y_j < y < y_{j+1}$. We return $('no', (\pi_y, Sign(y_j, y_{j+1}), (y_j, y_{j+1})))$.

Verification of answer: $Verify(x, q, b, \pi, PK)$: If $b =' yes'$ then parse π as a signature coupled with its content (σ, x, v) and verify that it is valid by checking that $Ver_{PK_{sig}}(\sigma, (x, v)) = 1$. If that is the case return 1, else return 0. If $b =' no'$ then parse π as $(\pi_y, \sigma, y_1, y_2)$ and verify that $Ver_{PK_{sig}}(\sigma, (y_1, y_2)) = 1$. If the signature is valid then check that $y_1 < h_2(\pi_y) < y_2$ and that $RSA_{PK_{rsa}}(\pi_y) = h_1(x)$. If this is the case return 1, else return 0.

Remark III.1. *Note that if a* resolver *learns DS it only knows the values of F over the set R, which it could also learn by sending random queries to a* secondary *until he gets all signatures $\{Sign(y_j, y_{j+1})\}_{j=0}^r$. On the other hand if a* resolver *learns SS it could exploit it to enumerate over R by validating all the signatures $\{Sign(x_j, v(x_j))\}_{j=1}^r$, or use SK_{rsa} to perform a dictionary attack like the ones attributed to NSEC3. Below is an illustration of I_S.*

[5]In order to find the correct signature we can either sort the signatures $\{Sign(x_j, v(x_j))\}_{j=1}^r$ by their values $F(x_j) = y_j$ and then know which signature matches each y_j or by adding to every signature the element x_j explicitly and have the appropriate lookup table.

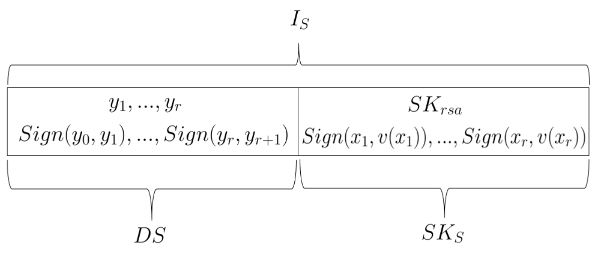
I_S
$y_1, ..., y_r$
$Sign(y_0, y_1), ..., Sign(y_r, y_{r+1})$
SK_{rsa}
$Sign(x_1, v(x_1)), ..., Sign(x_r, v(x_r))$
DS
SK_S

Fig. 2. Illustration of I_S.

A. Computational Requirements of NSEC5

The real computation cost of NSEC5 comes at the *secondary*, who needs to perform a single RSA signing computation plus two hash functions; however such "online signing" was already proposed in RFC 4470. The *resolver* needs to verify an RSA computation (forward) and verify a signature plus hash computation; is very similar to work needed for NSEC3, except that now two signatures must be verified instead of one, and recall that RSA signature verification is very fast. The setup algorithm by the *primary* requires $r = |R|$ computations of backward RSA and $2r + 1$ signatures; again this is not all that different than the requirements of NSEC3.

B. NSEC5 is a Good PSR

Theorem III.2. *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. First we notice that for every $x \in U$ there exists exactly one pair (y, π) for which it holds that $F(x) = y$, $h_1(x) = RSA_{PK_{rsa}}(\pi)$ and $h_2(\pi) = y$. This is true as h_1, h_2 and the RSA permutation are all deterministic algorithms and RSA is also a permutation.

The second property is verifiable pseudorandomness:

Lemma III.3. *For every $x \in U$ the value $F(x)$ is pseudorandom over $\{0, 1\}^n$ in the following sense: no 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 equal to x can distinguish $F(x)$ from a random value in $\{0, 1\}^n$.* [6]

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 as in Appendix C. Assume wlog that for every $x_i \neq x$ that A asks to evaluate $h_1(x_i)$ it also asks to see $(F(x_i), S(x_i))$ and that the upper bound on the number of queries is q. Given a public key (N, e) and challenge z that we wish to invert, before A's first query we draw uniformly at random $c_1, .., c_q \in [N]$

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

(where $[N]$ is the domain/range of the RSA permutation). We compute $z_i = RSA_{PK_{rsa}}(c_i) = c_i^e \bmod N$. Now every time A issues a new query x_i (on an element that wasn't queried before) we set $h_1(x_i) = z_i$ and that determines that $S(x_i) = c_i$ and $F(x_i) = h_2(c_i)$. When h_1 is queried on x we return z. When h_2 is queried we answer with a random and consistent (with previous answers) manner. When A queries h_2 on a point p we check whether $RSA_{PK_{rsa}}(p) = z$. If it is equal, we are successful in the inversion.

The distribution A witnesses is identical to the real distribution. There are two possible cases: If A didn't query h_2 on $S(x) = RSA^{-1}_{SK_{rsa}}(z)$, then it cannot distinguish between the two random values with greater than 0 advantage. If A queried h_2 on $S(x) = RSA^{-1}_{SK_{rsa}}(z)$ then we successfully managed to invert the RSA permutation over z. Thus, A's advantage in distinguishing the two values is the probability of successfully inverting. ■

The proof generalizes naturally from distinguishing the true value of F on a single element from a random value to distinguishing the true values of a whole set $R \subset U$ from a set of random values, (recall that the hardness assumption on inverting any of a set of r values is as hard as a single element C[7]). This helps us construct a simulator which can draw at random values in the range of F to represent the values of F on the set R, while keeping the two sets of values indistinguishable from one another.

In order to show that NSEC5 constitutes a PSR system we need to prove the following three properties (Definitions II.2, II.3 and II.4):

Completeness. For every $R \subseteq U$, $v : R \rightarrow V$ and $x \in U$, when we run $Setup(R, v(\cdot), 1^k) = (PK, I_S)$ and then run $Query(x, PK) = q$ we need to show that

$$\Pr[Verify(x, q, Answer(q, I_S, PK), PK) = 1] = 1 - \mu(k)$$

where $\mu(k)$ is a negligible function.

If $x \in R$, then by the way we defined the Setup algorithm there exists an index $j \in [r]$ for which $x_j = x$ and we generated

$$Sign(x_j, v(x_j)) = Sig_{SK_{sig}}(x_j, v(x_j))$$

so the Answer algorithm will find the signature $Sign(x_j, v(x_j))$ and it will be validated correctly by the verification algorithm with probability 1.

If $x \notin R$, then we claim that with overwhelming probability $F(x) \neq y_j$ for every $j \in [r]$. Otherwise, we could guess $F(x)$ with non-negligible probability without querying for $F(x)$ and this violates the pseudorandomness of F, proved in Lemma III.3, as long as $|R|/2^n$ is negligible. Furthermore an adversary could not even find an element $x \notin R$ to violate the completeness property with non-negligible probability, otherwise it will again contradict the lemma.

Soundness. First note that as we showed earlier, for every $x \in U$ there exists only one pair (y, π) for which it holds that $F(x) = y$, $h_1(x) = RSA_{PK_{rsa}}(\pi)$ and $h_2(\pi) = y$. Assume for contradiction that there exists some polynomial time adversary A that using (PK, I_S) can provide for some $x \in R$ a proof that $x \notin R$ with non-negligible probability. This means that this adversary A can forge a signature with non-negligible probability for a pair of fake values (y_1, y_2) where for at least one of them it holds that $y_i \neq y_j$ for every $j \in [r]$, as the original signatures $\{Sign(y_j, y_{j+1})\}_{j=0}^r$ cannot provide the false proof for $(F(x), S(x))$. According to the security assumption of the signature scheme one cannot forge signatures on *any* message not signed by the signer with non-negligible probability.

In case A can provide a valid proof for some $x \notin R$ that proves that $x \in R$ with non-negligible probability, then it means A can forge signatures for (x, v) with non-negligible probability, again, violating the security property of the signature scheme.

Note that it holds that an adversary cannot even find an element $x \in U$ to violate the soundness property with a non-negligible probability as this means it can forge a signature of his choosing, violating the security property of the signature scheme.

Privacy. In order to show that for $f(R) = |R|$ the system NSEC5 is f-zk we need to show a suitable simulator SIM, where no probabilistic polynomial time adversary can distinguish an interaction with the real system and SIM.

On its first step of the computation $SIM^R(1^k, 1^{|R|})$ runs the RSA setup algorithm and obtains (PK_{RSA}, SK_{RSA}) and also runs the setup algorithm of the signature scheme and obtains (PK_{sig}, SK_{sig}). SIM then chooses the random oracles h_1, h_2 as in the setup algorithm of the PSR. SIM randomly selects $|R|$ values out of F's range and sorts them lexicographically, $y_1, ..., y_r \in \{0,1\}^n$ and creates the signatures $\{Sign(y_j, y_{j+1}) = Sig_{SK_{sig}}(y_j, y_{j+1})\}$ where we add the end points $y_0 = 0^n$ and $y_{r+1} = 1^n$ as the Setup algorithm does.

The simulator then outputs $PK^* = (PK_{rsa}, PK_{sig}, h_1, h_2)$ and a fake simulator key

$$SK^*_{SIM} = (SK_{rsa}, SK_{sig}, \{Sign(y_j, y_{j+1})\}_{j=0}^r, \{y_j\}_{j=1}^r),$$

which we can see is very similar to the original parameters I_S that the *secondary* usually gets but it is missing the signatures $\{Sign(x_j, v(x_j))\}_{j=1}^r$ and has the secret key for the signature scheme instead.

On its next rounds the simulator does the following: for each query it receives q_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 (remember $q_i = x_i$). If $x_i \in R$ then SIM generates a new signature $s_{x_i} = Sig_{SK_{sig}}(x_i, v_i)$ and returns $('yes', s_{x_i}, (x_i, v_i))$. Because the signer produces consistent signatures on the same query we will always get the same signature on the same message, i.e. $s_{x_i} = Sign(x_i, v(x_i))$. If $x_i \notin R$ the simulator computes $(F(x_i), S(x_i)) = (y_{x_i}, \pi_{x_i})$ and searches in SK^*_{SIM} for a j for which $y_j < y_{x_i} < y_{j+1}$. If we find such a j we return $('no', \pi_{x_i}, Sign(y_j, y_{j+1}), (y_j, y_{j+1}))$. If we don't find such a j, *i.e.,* a collision has occurred, we abort as we fail to produce an indistinguishable view.

Now we need to show that the view of the adversary communicating with the simulator is indistinguishable from

[7]This is one place where the specific properties of RSA are used, rather than a generic trapdoor permutation, where we would have to loose a factor r in the advantage due to hybrid argument

that of the adversary communicating with the real system. The public key PK^* is generated by the same algorithms the real system uses. Proofs regarding $x \in R$ are signatures $Sig_{SK_{sig}}(x, v(x))$, generated the same way the original proofs are created in the system, the only difference is that they are generated online instead of before hand during the setup phase, but this yields the same distribution. The only difference the adversary witnesses is that instead of real values of F on points of R it gets random values. However, we argued in Lemma III.3, that a polynomial time adversary cannot distinguish between $\{F(x_i)|x_i \in R\}$ and a collection of $|R|$ random values in $\{0,1\}^n$ with more than a negligible advantage. Thus, it cannot distinguish the simulation from a real execution. ■

IV. Cryptographic Lower Bounds

The point of this section is to show that the *secondary* in a PSR system must perform a somewhat non-trivial computational task (public key computation) rather then hashing *on each query*. This is done by showing how to obtain public-key authentication (PKA) and identification protocols from PSR Systems where the complexity of the prover or authenticator is similar to that of the *secondary*. Thus if we can construct public key authentication and identification protocols using PSR systems then the task of constructing such protocols cannot be harder than that of constructing PSR systems. Those protocols are not known to have any implementations which are much more efficient than signature schemes, which is why we can conclude that a non-trivial computational task is required to construct PSR systems.

Public-key authentication can be seen as a relaxation of signature schemes where we give up the *transferability* property, *i.e.,* that the receiver of the signature can convince a third party that the signature is valid rather than just convince himself. In a public-key authentication system (see [DDN00] Section 3.5) the prover is the owner of the public-key and can engage in an *interactive* protocol (rather than a single message as in signatures) with the verifier and convince the latter that the owner of the public-key is indeed approving the message.

In *identification* protocols there isn't even a message; instead the prover convinces the verifier that he is alive. For example we could use a public-key identification protocol in key cards, where each card opens the doors which that specific employee is allowed to open. The door should not know the secret key of the key card, just its public-key, so that if it is broken into the damage would be limited. The card will be a prover and the door a verifier for the identity of the card holder. 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. Public-key authentication is generally harder than identification, since the verifier can challenge the prover with a random message and ask to authenticate it.

A. Public-key Authentication Security

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* $(Setup, Prove, Verify)$ *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 A wins the game where it selects* M *only after the attack with probability at most* ε.

B. PKA from PSR

We show how we can use a PSR system (PSR_Setup, Q, A, V) which is selectively secure against polynomial time adversaries (as in Definition II.5) and construct a Public key authentication protocol

$$(PKA\_Setup, Prove, Verify)$$

which 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 for the PSR; $PSR\_Setup(R, v(\cdot), 1^k)$ and obtain (PK, I_S) which will be our public and secret keys.
- $Prove(M_i, I_S, PK)$: The prover will act as the *secondary* in the PSR system proving that $M_i \notin R$.
- $Verify(M_i, PK)$: The verifier acts as the *resolver* in the PSR system and accepts if the *resolver* accepts the proof of non-membership.

Remark IV.3. Note that the way we have defined the authenticator does not satisfy perfect completeness (if the verifier happens to choose M_R we cannot authenticate that message). We can get back 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* (PSR_Setup, Q, A, V) *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

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

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. ■

C. 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). We do that because we want our lower bound to be as tight as possible and as existential security is a stronger requirement then selective security it reduce PSR systems to a harder problem, which as we will see next we can get for free in the random oracle model. 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.

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 $\varepsilon' = \frac{\epsilon}{q(k)}$. ■

D. On Signatures and Transferability

We have seen that PSR systems can be used to construct both public key authentication schemes and identification schemes. Our goal in this section is to point out that for many PSR systems we can actually get a signature scheme. This benefits us as signature schemes are obviously a much stronger primitive than identification and authentication schemes as a signature scheme implements both primitives naturally. Thus we show a strong relation between PSR systems and signature schemes, which are used in our NSEC5 protocol, thus proving that signatures are essential to constructing PSR systems.

We divide PSR systems into two types by the way their Query algorithms work:

Deterministic:
: Doesn't use any random coins. Our NSEC5 construction has this property since $Query(x, PK) = q = x$. Also as mentioned before, in DNSSEC queries are sent in the clear which achieves higher performance by avoiding precomputation of any sort.

Randomized in the public coins model:
: an algorithm which uses randomness, but doesn't generate any secret information. Given the answer, a *resolver* doesn't need any prior knowledge on the query issued to validate the response, it is clear whether it is accepted or not.

Note that in all cases, by definition, x can be deduced efficiently from the query $Query(x, PK)$.

We claim that, in the random oracle model, a PSR system with either type of queries can be used to construct an existentially secure signature scheme. This is done using the Fiat and Shamir [FS86] transformation which constructs signature schemes from identification schemes. If we use the public key authentication protocol which is constructed using a PSR system in Section IV then we can get a selectively secure public key authentication protocol using Theorem IV.4. In the random oracle model we also proved that we can use this protocol to make it existentially secure as in Theorem IV.5.

We now show that we can get a signature scheme from the two different variants of PSR systems described above. If we are in the deterministic case (the DNSSEC case as well) then a user who wishes to validate the signature

$$Answer(Query(x, PK), I_S, PK) = (b, \pi)$$

only needs the public key PK and to use the Query algorithm to calculate $Query(x, PK) = q$ and then the user can check that $Verify(x, q, b, \pi, PK) = 1$ which validates the signature. It is obviously transferable as there is no secret information in generating the query as it is a deterministic algorithm. In the public coins model we can simply do the following. Define a random oracle h which may be part of the public-key. The random bits for the algorithm $Query(x, PK)$ will be $h(x)$, thus making this algorithm also deterministic and we get a signature scheme again.

E. 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-B. 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.

References

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

[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*, EUROCRYPT, 2005, 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.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham, *Short signatures from the weil pairing*, J. Cryptology **17** (2004), no. 4, 297–319.

[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*, EUROCRYPT, 2005, 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.

[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.

[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.

[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.

[TT10] Roberto Tamassia and Nikos Triandopoulos, *Certification and authentication of data structures*, AMW, CEUR Workshop Proceedings, vol. 619, CEUR-WS.org, 2010.

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 for that. 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 $m \in [N]$ and the public key and computes $RSA_{PK_{rsa}}(m) \equiv m^e \bmod N \equiv \sigma \bmod N$. The RSA backword algorithm takes a value $\sigma \in [N]$ 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 $x \in [N]$ is mapped by the encryption algorithm to some unique $y \in [N]$ and the decryption algorithm maps y back to x.

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 $y \in [N]$, it holds that

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

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 $w_i \in [N]$ 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.