Primary-Secondary-Resolver Membership Proof Systems

Moni Naor* and Asaf Ziv

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

Abstract. We consider *Primary-Secondary-Resolver Membership Proof Systems* (PSR for short) and show different constructions of that primitive. A PSR system is a 3-party protocol, where we have a *primary*, which is a trusted party which commits to a set of members and their values, then generates public and secret keys in order for *secondaries* (provers with knowledge of both keys) and *resolvers* (verifiers who only know the public key) to engage in interactive proof sessions regarding elements in the universe and their values. The motivation for such systems is for constructing a secure Domain Name System (DNSSEC) that does not reveal any unnecessary information to its clients.

We require our systems to be complete, so honest executions will result in correct conclusions by the *resolvers*, sound, so malicious *secondaries* cannot cheat *resolvers*, and zero-knowledge, so *resolvers* will not learn additional information about elements they did not query explicitly. Providing proofs of membership is easy, as the *primary* can simply precompute signatures over all the members of the set. Providing proofs of non-membership, i.e. a denial-of-existence mechanism, is trickier and is the main issue in constructing PSR systems.

The construction we present in this paper uses a set of cryptographic keys for all elements of the universe which are not members, which we implement using hierarchical identity based encryption. In the full version of this paper we present a full analysis for two additional strategies to construct a denial of existence mechanism. One which uses cuckoo hashing with a stash, where in order to prove non-membership, a secondary must prove that a search for an element will fail. Another strategy uses a verifiable "random looking" function and proves non-membership by proving an element's value is between two consecutive values of members.

For all three constructions we suggest fairly efficient implementations, of order comparable to other public-key operations such as signatures and encryption. The first approach offers perfect ZK and does not reveal the size of the set in question, the second can be implemented based on very solid cryptographic assumptions and uses the unique structure of cuckoo hashing, while the last technique has the potential to be highly efficient, if one could construct an efficient and secure VRF/VUF or if one is willing to live in the random oracle model.

* 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.

1 Introduction

We consider the cryptographic primitive called *Primary-Secondary-Resolver Membership Proof Systems* (PSR for short) and show efficient constructions of that primitive. The motivation for this type of systems comes from trying to improve DNSSEC which is a security extension of DNS (Domain Name System) (plain DNS communication doesn't guarantee security (confidentiality and authenticity) for the users). The basic problem is as follows, we have a trustworthy source, called the *primary*, which maps all valid names (e.g. URLs) in its domain to their corresponding values (e.g. IP addresses). This *primary* doesn't communicate directly with users (*resolvers*) who wish to make DNS queries for names; it has the *secondaries* for that, which are DNS servers that receive some initial information from the *primary* and are in charge of responding to *resolvers*' queries. As there may be many such *secondary* servers, we cannot be sure they are all honest and we do not wish to give them the ability to fool *resolvers* with a false response to a DNS query. We would like to give them enough information so as to give correct responses to DNS queries and a short proof of some sort to help convince the *resolver* of the authenticity of the data they received. On the other hand, we do not wish the *resolvers* get more information about the domain than a simple answer to their query, i.e. whether the answer is positive or negative is all a *resolver* should be able to deduce (the issue of releasing too much information about the domain has been an obstacle in getting the current DNSSEC adapted [4]).

A PSR system consists of a setup algorithm, used by the *primary* which receives a privileged subset R from a universe U of names (e.g. the list of hosts in its domain) and a set of corresponding values V, mapping each element $x_i \in R$ to its value $v_i \in V$ (e.g. mapping all URLs in a domain to their IPs). The *primary* generates a public key PK (one may think of it as a signature key), which should be available to all parties of the protocol. It also generates a secret key SK which provides *secondaries* the ability to answer queries honestly. We will be interested only in *efficient* constructions where the public key size and the amount of communication between the *secondaries* and the *resolvers* are independent of the cardinality of the set R.

1.1 Our Contributions

In a companion paper to this work [19] the notion of PSR systems was introduced (albeit it was defined as a one-round proof protocol), as well as an efficient construction named NSEC5 was suggested. NSEC5 is based on RSA and analyzed in the random oracle model. The main application of PSR systems is for a secure Domain Name Server that does not reveal information about the underlying set. That paper also gave a lower bound that shows that in order to preserve soundness and prevent an adversarial *resolver* from learning additional information about elements they didn't query, the *secondary* must perform some non-trivial computation: it must do the computational work needed in a a public key identification scheme, for which the best known implementations are signa-

tures (in the random oracle model these two are equivalent). (This showed that none of the prior approaches to DNSSEC such as NSEC3 yield a solution that is secure against zone enumeration, i.e. listing of the set R).

We consider PSR Systems that are more general than those of [19] and define PSR systems with interactive proofs as well as systems that are *perfect* zero-knowledge.

In this paper we investigate in depth PSR systems. Our main interest is efficiency, where we are interested in the computational and communication load on all three parties, but in particular in the secondary-resolver part that is performed online. Our main goal in this work is to provide PSR systems that are efficient and based on reasonable and well studied assumptions. We aim for efficiency that is of the order of other public-key primitives such as encryption and signatures.

We provide three general techniques to constructing PSR systems and present efficient implementations to each of them. We use signatures and various different cryptographic primitives in our constructions such as: hierarchical identity based encryption schemes, one-time signatures, cuckoo hashing (with a stash) with commitments and fixed-set non-membership proofs, verifiable random/unpredictable functions and pseudorandom functions with interactive zero-knowledge proofs. Our constructions are based on solid cryptographic assumptions: the discrete logarithm assumption and factoring, the existence of universal one way hash functions and various Diffie-Hellman assumptions. Some of our constructions even achieve perfect zero-knowledge.

It is quite clear that the more challenging case in constructing PSR systems is dealing with the *non*-members of the set. For the members of the set a precomputed signature by the *primary* solves the problem. We suggest three approaches for constructing PSR systems. All constructions use (regular) signatures to handle proofs of membership, as we precompute a signature over every $x_i \in R$ and its value v_i. Thus, the difference between the constructions is how they handle proofs of non-membership, i.e. we offer different denial of existence mechanisms.

In our first approach the *primary* matches encryption keys to elements of the universe U. A *secondary* with knowledge of such a key can use it to generate a proof of non-membership for the corresponding element. The *primary* precomputes a set of secret keys K, from which it can derive the keys corresponding *only* to the set of elements $U \backslash R$ and sends it to the *secondaries* as part of their secret key. As long as we make sure the *secondaries* cannot produce any key for an element in R, we can construct a denial of existence mechanism in a number of ways. *Resolvers* can encrypt a random challenge, which can be decrypted only with the secret key corresponding to the queried element $x \in U$, thus non-membership can be proven only for elements outside of R. One can also just send that secret key to *resolvers* when queried, making them verify the correctness of the key by encrypting and decrypting random challenges by themselves. The *secondaries* can also generate signatures for the queried element under a secret key corresponding to that element and verified with a corresponding public key. In order to implement those constructions efficiently we use *Hierarchical Identity*

Based Encryption (or HIBE for short). One can think of a set of identities as nodes in a full binary tree, where with the secret key for an identity, one can produce the key to any of its descendants. We think of the leaves as elements in the universe, so by making sure the set of keys K doesn't contain any secret key to an element in R or any of its ancestors, but contains at least an ancestor key to the rest of the elements in U, we get an efficient denial of existence mechanism. Lastly we consider a construction that uses a chain of signatures from the root of the tree to the leaf, where each signature signs the public key needed to verify the next signature in the chain. All those constructions manage to achieve *perfect* zero-knowledge.

The idea of the second approach is to imitate an oblivious search for the element, where by oblivious we mean that the locations examined are determined by the element searched and some hash functions. The point is to show that the searched element is in none of the probed locations. For the data structure we use cuckoo hashing [36] where (unless we are unlucky) each element resides in one of two locations. That is, as a denial of existence mechanism, we need to prove non equality just twice. To handle the unlucky case we use a cuckoo hashing scheme with a stash [26] to store some extra elements. We need to prove non equality to these elements as well, however we have the advantage that these elements are fixed for all possible searches. To handle the "normal" case the *primary* places Pedersen commitments [37] for the relevant elements in the cells of the cuckoo hash tables (including "dummies" in the empty cells) and signs these commitments. The *secondary* is provided with the signed commitments and proves the committed values are not equal to the queried element. For the stash non equality we use a generalization of the Feige-Fiat-Shamir identification protocol [15]. Both proofs are zero knowledge and are rather efficient as the computation needed in order to execute these two interactive zero-knowledge protocols is dominated by only a constant number of exponentiations. As Pedersen commitments rely on the discrete logarithm assumption and the Feige-Fiat-Shamir protocol relies on the factoring assumption, the result is a PSR system which reveals the size of the set R but is very efficient and is based on conservative and well studied cryptographic assumptions.

Our third approach to constructing PSR systems applies a "random looking" function F, for which we can prove the value $F(x)$ in a zero knowledge fashion, without revealing information about the value of the function at other locations. The *primary* precomputes the values of F over the set R, sorts them lexicographically and signs them in pairs, $\{Sign(y_i, y_{i+1})\}_{i=0}^{r}$. In order to prove non-membership for an element $x \notin R$ one simply provides a proof that $F(x) = y$ and the signature $Sign(y_i, y_{i+1})$ for which $y_i < y < y_{i+1}$ (we choose F to have negligible probability for collisions). This construction reveals the size of the set R during multiple executions of the protocol as a *resolver* which issues enough random queries will eventually witness all signatures $Sign(y_i, y_{i+1})$ and learn the size of R, but in some applications such as DNSSEC, revealing the size of the set is acceptable. In order to construct the function F we use variants of *Verifiable Random Functions* (VRF) and *Verifiable Unpredictable Functions* (VUF) [30],

the Naor-Reingold PRF [32] with zero knowledge interactive proofs, the GHR signature scheme [16] and a random oracle construction which uses the famous BLS signature scheme [7]. The scheme NSEC5 presented in [19] (which resides in the random oracle model) falls into this category as well.

For all three constructions we suggest fairly efficient implementations. The first approach offers perfect ZK and does not reveal the size of the set in question, the second can be implemented based on very solid cryptographic assumptions and uses the unique structure of cuckoo hashing, while the last technique has the potential to be highly efficient, if one could construct an efficient and secure VRF/VUF or one is willing to live in the random oracle model.

Structural Issues: We analyze and prove that PSR systems with one-round proofs are secure even in a concurrent setting. This means that in the case of one-round proofs, even a coordinated attack of *resolvers* trying to learn information about elements in the universe which they did not query explicitly will fail with overwhelming probability. In the case of many-rounds proofs we show that providing each *secondary* with an independent set of keys also results in a concurrently secure PSR system. We prove that PSR systems exist if and only if one way functions exist, which in turn helps us get a black box separation from *zero knowledge sets* [29], which is a more restrictive membership proving system (see details in Section 1.3), thus showing that the two primitives are indeed inherently different.

1.2 A Guide for Reading the Paper

In Section 2 we present our model, the definition of PSR systems, our requirements of completeness, soundness and zero-knowledge and in Section 3 we show cases where the system is secure in a concurrent setting. In Section 4 we show a HIBE based construction which achieves perfect ZK. In Section 5 we give a short description and intuition regrading our cuckoo hashing with a stash based PSR and the one based on "random looking" functions (the full version [34] gives a more detailed description). We also present a signature based PSR system and use it to prove that the existence of one way functions is equivalent to the existence of PSR systems, which leads us to a black box separation between PSR systems and ZKS [29]. In Section 6 we present concluding remarks.

1.3 Related Work

There are several types of cryptographic primitives that are related to PSR systems. Consider *zero-knowledge sets*, introduced by Micali, Rabin and Kilian [29] (ZKS for short) and its generalization zero-knowledge elementary databases. The latter is a primitive, defined in the common reference string model or the trusted parameters model, where a user (prover) can commit to a database and later open and prove its values to a verifier in a zero knowledge fashion. The existence of ZKS implies the existence of a PSR system, as a zero-knowledge elementary database construction implements a PSR System (the other direction is not true

as implied by Corollary 2). However, the problem is that even the best known constructions of ZKS are inefficient. The point is that in a ZKS the requirements are too stringent: even the *primary* cannot cheat. This is not something of interest in our setting, since the *primary* is a trustworthy party that commits to a set of its choosing and it does not make sense for it to cheat. We are only interested in preventing the *secondaries* from cheating. Hence we introduced a more complex setting with three parties, at the benefit of gaining efficiency.

Chase et al. [12] introduce the notion of trapdoor mercurial commitments (TMC for short) and construct ZKS based on TMCs. They show a few implementations of their new primitive where their most efficient implementation is a constant factor improvement on the original MRK construction, while both rely on the discrete logarithm assumption. Catalano et al. [11] extend their notion of TMC to trapdoor q-mercurial commitments (q-TMC for short) and by that further improve the efficiency of ZKS implementation by shortening the non-memberships proofs by a constant factor, at the expense of slowing down the verification process. Their construction of q-TMC relies on the q-strong Diffie-Hellman assumption. Later, Libert and Yung [28] introduced a new construction for q-TMCs, based on the q-Diffie Hellman exponent assumption, and managed to shorten the memberships proofs by a constant factor as well. All those ZKS constructions have the same basic structure: a tree (either binary as in [29,12] or with arity q as in [11,28]), where the leaves represent the elements in the universe and a proof of membership or non-membership is a path of commitments from the root to the leaf. All four ZKS constructions use proofs made up of $O(\log |U|)$ group elements and require $O(\log |U|)$ modular exponentiations for verification.

Prabhakaran and Xue introduced *statistically hiding sets* [38] (SHS for short), which are a slight variation on ZKS. Their definition of statistical hiding is formulated with computationally unbounded simulation, which means it is a relaxation of the security requirement of ZKS as they do not require efficient simulation. Their construction uses accumulators, first presented in [5], in order to accumulate a set of values into one value, where there is a short proof for every value in the set. Although it is more efficient than ZKS and can be extended to statistical hiding databases, their underlying assumptions are rather new and strong. They use the strong RSA assumption and an assumption they call the knowledge of exponent assumption. They require the use of a hash function which maps elements to large prime numbers and a trapdoor DDH group.

Ostrovsky, Rackoff and Smith [35] generalized ZKS by defining Consistent Query Protocols, which allow more general queries than membership queries. They also suggested a relaxation for ZK proofs, allowing the server to leak an upper bound T on the size of the database (called size-T-Privacy). Our privacy requirement, f-ZK, is a generalization of this size-T-Privacy requirement.

Another related line of investigation is that of data structures that come with a guarantee of correctness. That is when the data structure, like a dictionary, returns an answer it also provides a proof that the answer is correct in the sense that it is consistent with some external information. One motivation for these investigations comes from data structure for managing CRLs (certificate

revocation lists). The difference with the current work is that no additional information than the result of the query should leak.

A recent paper by Ghosh, Ohrimenko and Tamassia [18] introduces two new primitives which are related notions to PSR systems: a 2-party and a 3-party protocols for proving values of elements in a database and their order (lists). The 2-party protocol they define is Zero Knowledge Lists (ZKL for short), where their construction of the primitive is too inefficient for our needs, as it builds upon ZKS (which, as we mentioned, does not have an efficient implementation yet). The 3-party protocol is Privacy Preserving Authenticated Lists (PPAL) which unlike ZKL is closer in spirit to our PSR systems but it cannot answer non-membership queries (their construction only handles queries for elements in the list and returns their order in the list combined with a proof). Besides that, their constructions are also analyzed in the random oracle model, where we strive to find constructions in the standard model.

2 Model and Security Definitions

We model Primary-Secondary-Resolver Membership Proof systems as a 3-party protocol where the *primary*, a trusted party, commits to a set R, a subset of the universe U, where each element $x_i \in R$ is coupled with a value $v_i \in V$. The *primary* generates two keys for the committed set, the secret key SK given only to *secondaries* in the system and the public key PK given to all parties of the protocol, i.e. *secondaries* and *resolvers*. The *resolvers* in the system engage in an interactive protocol with the *secondaries* in order to learn whether a given $x \in U$ is in R or not and if yes then they obtain its value v_x. The *secondaries* use their secret key to generate proofs (possibly interactive) for the correct statement regarding the queried element, while *resolvers* verify the correctness of the proofs they get. We require that the *secondaries* won't be able to cheat the *resolvers* and if the *secondaries* are following the protocols then the *resolvers* should be able to verify the correctness of the responses with overwhelming success probability. Another important requirement we would like from such a system is zero-knowledge, i.e. for *resolvers* to learn as little as possible about elements they didn't query explicitly. See Figure 1 for an illustration of the 3-parties' engagement in the protocol.

Remark 1. Note that we chose to focus on the static version of this problem, i.e. when the sets R and V are determined at the beginning of the process and do not change throughout the process. The dynamic case for this problem is out of the scope of this paper, though we discuss the issues of defining requirements for the dynamic case, as well as give guidelines on how to transform our constructions into ones which can handle dynamic changes in the full version of the paper [34].

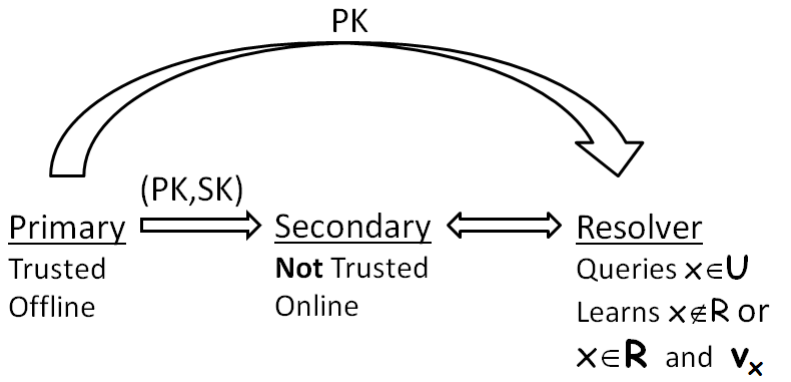

Fig. 1. Illustration of a PSR system.

2.1 PSR Systems

The system consists of three 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 secret key SK, delivered to the *secondaries*. The *resolvers* use the *Verify* algorithm in order to initiate an interactive proof session with the *secondaries* who use the *Prove* algorithm to prove interactively the correct membership statement about the element, queried by the *resolver*.

Definition 1. *Let U be a universe of elements. A Primary-Secondary-Resolver system (PSR for short) is specified by three probabilistic polynomial-time algorithms $(Setup, Prove, Verify)$:*

$Setup(R, V, 1^k)$**:** *On input k the security parameter, a privileged set $R \subseteq U$ and its values V, where $|R| = |V| = r$ (for every $x_i \in R$ the corresponding value is $v_i \in V$), this algorithm outputs two strings: (PK, SK) which are the public and secret keys for the system.*

$Verify(x, PK)$**:** *The algorithm gets as input $x \in U$ and the public key PK. It initiates an interactive proof protocol over the element $x \in U$ with a* secondary *of its choice and verifies the correctness of the proof given by the* secondary*. It outputs* 1 *when it accepts the interactive proof and* 0 *otherwise.*

$Prove(x, PK, SK)$**:** *On input $x \in U$ and both the public and secret keys (PK, SK) this algorithm proves interactively to a* resolver *either the statement $x \in R$ and its value is v_x or $x \notin R$.*

We require the above three algorithms to satisfy three properties: completeness, soundness and zero knowledge.

2.2 Completeness and Soundness

The *completeness* requirement means that when the parties at hand are honest and follow the protocol, then the system works properly. The *resolvers* will learn successfully whether the element $x \in U$, which they queried, is in R (and its value) or not. We do allow a *negligible* probability of failure.

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

$$\Pr\left[\begin{array}{l}(PK, SK) \stackrel{R}{\leftarrow} Setup(R, V, 1^k);\\ Verify(x, PK) \stackrel{R}{\leftrightarrow} Prove(x, PK, SK);\\ Verify(x, PK) = 1\end{array}\right] \geq 1 - \mu(k)$$

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

As for *soundness*, we want that even a malicious *secondary* A, would not be able to convince an honest *resolver* of a false statement with more than a negligible probability. We require this to hold even when the adversary gets to choose R and V, then gets the keys (PK, SK) and then chooses $x \in U$ on which it wishes to cheat. At the end of the protocol A outputs either 0 if it tries to convince the *resolver* that $x \notin R$ or $(1, v)$ if it tries to convince him that $x \in R$ and its value is v.

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

$$\Pr\left[\begin{array}{l}(R, V) \stackrel{R}{\leftarrow} A(1^k);\\ (PK, SK) \stackrel{R}{\leftarrow} Setup(R, V, 1^k);\\ x \stackrel{R}{\leftarrow} A(PK, SK);\\ Verify(x, PK) \stackrel{R}{\leftrightarrow} A(x, PK, SK);\\ Verify(x, PK) = 1 \wedge\\ ((A(x, PK, SK) = 0 \wedge x \in R) \vee\\ (A(x, PK, SK) = (1, v) \wedge (x \notin R \vee (x = x_i \wedge v \neq v_i))))\end{array}\right] \leq \mu(k)$$

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

Note that our definitions are strong because they ensure (up to negligible probability) that an adversary cannot find any $x \in U$ violating either completeness or soundness, even after getting its relevant keys, i.e. (PK, SK) for a *secondary* in the soundness condition and PK for a *resolver* in the completeness condition.

2.3 Zero-Knowledge

We want to restrict the amount of information learned about the set R by *resolvers* during the interactive proofs. Besides the answer to the question being asked by the *resolver* we would like him to learn as little as possible about the

set R. In some cases we let some information about the set R leak during the protocol (or many executions of the protocol on different elements), which is why we choose to define zero-knowledge with respect to a function f acting on R. We show two constructions of PSR systems which don't leak any information about the set R (see Sections 4 and 5.1), while the rest of the constructions leak the size of the set R (see Section 5) We define this property as **f-Zero-Knowledge** (f-ZK for short), where $f(R)$ is some information about the set which we can tolerate leaking to *resolvers*.

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 query asked by the *resolver* online, but must still be able to "forge" a satisfactory proof for that response. This allows us to deduce that the *resolver* has not learned much about R from the proofs, for if it had, it would be able to distinguish between an interaction with the simulator and one with the real *secondary* (at least after it gets R explicitly).

The PSR simulator: Let SIM be an interactive polynomial time algorithm with restricted oracle access to the set R, which means it can query the oracle only on elements which the adversary communicating with him queried explicitly. On its first step SIM receives $f(R)$ and outputs a fake public key PK^*, a fake secret key SK_{SIM} and $f(R)$. On its following steps an adversary interacts with the simulator and queries different elements in the universe. Following every such query x_i the simulator queries its oracle for x_i and either learns $x_i \notin R$ or $x_i \in R$ and its value is v_i. SIM proves interactively the statement on x_i to the adversary. The simulator is successful if its output, i.e. its random tape, public key and transcripts of the interactive protocols, is indistinguishable from that of a real PSR system.

The first step of the interactive protocol for the PSR system[1] is:

$$(PK, SK, f(R)) \overset{R}{\leftarrow} Setup(R, V, 1^k)$$

and for the simulator the first step is:

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

The rest is a series of interactive proofs of membership between the adversary and either a PSR system or a simulator, where the simulator uses the fake public key PK^* and the fake secret key SK_{SIM} to respond to queries and the system uses the real keys (PK, SK).

Definition 4. *Let $f()$ be some function from 2^U to some domain and let algorithms $(Setup, Prove, Verify)$ 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 $\mu(k)$:*

[1] Note that the Setup algorithm is not defined to output $f(R)$, but it is obviously a simple modification, as it gets R and can compute $f(R)$ easily. We add this output in order to generate comparable views.

There exists a simulator SIM such that for every probabilistic polynomial time algorithms A (adversary) and D (distinguisher), a set $R \subseteq U$ and V, the distinguisher D cannot distinguish (See Remark 2 below) between the following two views (interactions of A with a PSR system or a PSR simulator) with an advantage greater than $\mu(k)$, even for D that knows R:

$$view^{real} = \{r_{real}, PK, f(R), (x_1, \pi_1), (x_2, \pi_2), \ldots\}$$

and

$$view^{SIM} = \{r_{SIM}, PK^*, f(R), (x_1, \pi_1^*), (x_2, \pi_2^*), \ldots\}$$

where the two views are generated by the protocols described above, π_i and π_i^* are the transcripts for the interactive protocols over the element x_i and r_{SIM} and r_{real} are the random tapes of the simulator and secondaries *respectively.*

Remark 2. We have three notions of Zero-knowledge for PSR systems: computational ZK, which means that the distinguisher cannot *computationally* distinguish between the two views, *statistical* ZK, where the distributions of the two views are statistically close and *perfect* ZK where the two distributions are identical. Note that the perfect and statistical ZK have the added advantage of being secure in an information theoretic sense, which guarantees *everlasting privacy*. As both these ZK properties are information theoretic, they require their underlying assumptions to hold *only during the execution of the protocol*, while for computational ZK, we require the assumptions to hold 'forever' in order to prevent an adversary from breaking the privacy of the scheme at a later point in time. Our HIBE and signature based constructions (Section 4 and 5.1 respectively) achieve perfect ZK, the cuckoo hashing construction (Section 5.2) achieves statistical ZK, while the last construction (Section 5.3) achieves computational ZK.

In our companion paper [19], we prove two very important facts about non-interactive PSR systems. The first is that f-ZK, where $f(R)$ is the cardinality of the set R, implies prevention of zone enumeration, i.e. if a PSR is f-ZK, then a *resolver* cannot learn any information about an element it didn't query explicitly. All of the constructions in this paper are at least f-ZK for this f (the HIBE and signature based constrictions are even perfect ZK), which means they all prevent zone enumeration. The second important result is that PSR systems require a heavy computational task from the *secondaries*, such as public key cryptography or public key authentication, in order to maintain both soundness and f-ZK. This fact is crucial to understanding why the *secondaries* work hard in our constructions. Note that both these proofs were for the single-round PSR and in the random oracle model, but the proofs generalize to our (possibly interactive) setting as well. The prevention of zone enumeration holds as is in the standard model for interactive proofs, while the reduction to public key authentication for interactive PSRs in the standard model is only selectively secure, as opposed to existentially secure in the random oracle model. We state the resulting theorem:

Theorem 1. *Given an f-ZK PSR system (where $f(R) = |R|$ or $f(R) = null$), one can construct a public-key identification or a selectively secure public key authentication protocol from the PSR system where the prover's complexity is similar to the* secondary*'s. The construction is black box.*[2].

3 Concurrent Zero Knowledge

In this section we prove that in some cases PSR systems are not only f-ZK as defined earlier, but also concurrent zero knowledge with respect to that same function f. Concurrent ZK was introduced by Dwork, Naor and Sahai [14] as an extension to zero knowledge. In order for an interactive proof system to be concurrent ZK we require that if we have up to a polynomial number of provers and verifiers, where the verifiers are controlled by a malicious adversary and work concurrently (one could start an interactive proof with a prover, put it on hold and finish an earlier interaction), then still no information is leaked to the adversary controlling the verifiers.

We use similar definitions to the ones defined by Rosen [40] and adapt them to our setting. For an interactive proof system $\langle P, V \rangle$, we define a nonuniform probabilistic polynomial time concurrent adversary A. A gets some input I (for PSR systems $I = PK$), controls a polynomial number of verifiers and has access to an unbounded number of copies of the prover P. A can use verifiers to interact with the provers and controls the scheduling of all the messages in the system, meaning that A controls when any verifiers output a message and when every prover outputs a message. We denote by $view_A^P(I)$ the view of the adversary, which is a random variable which contains the random tape of A and all the concurrent interaction of A with the provers (copies of P).

Roughly speaking, a protocol is concurrent ZK if for every such adversary A there is a probabilistic polynomial time simulator S_A such that the two ensembles $\{view_A^P(I)\}$ and $\{S_A(I)\}$ are computationally indistinguishable, where I is some $x \in L$ and $S_A(I)$ is the output of a simulator which uses the adversary A as an oracle. But PSR systems, as we defined them, consist of multiple executions of membership/non-membership interactive proofs using the keys (PK, SK). Thus it is more natural for us to define $I = PK$ and compare between the view of an adversary communicating with *secondaries* (provers) on the public key PK and the view of an adversary communicating with the simulator on the fake public key PK^*.

Thus we define a **concurrent PSR simulator** as a probabilistic polynomial time algorithm SIM, with restricted oracle access to the set R, such that on its first step of the computation, SIM gets $f(R)$ and outputs a fake public key PK^*, a fake secret key SK_{SIM} and $f(R)$. SIM is not allowed to query its oracle on $x \in U$ if it was not explicitly queried by a *resolver* (verifier) on it. When an adversary interacts with a simulator, the copies of the prover are replaced with the simulator itself which acts as a prover (i.e. it emulates all the provers), uses

[2] See the original paper for the proof and definitions for public key authentication.

the fake cryptographic keys it generated and can query its oracle for the element queried by the *resolvers*.

We consider two different concurrent settings: where all the *secondaries* get the exact same pair of keys and when each *secondary* and *resolver* get a pair of keys generated independently for them. We prove, that in the case we use independent keys, every PSR system which is f-ZK in the sequential (regular) setting is also f-CZK, thus by making the *primary* work $k \cdot m$ times harder, one can get a concurrently secure PSR system with k *secondaries* and m *resolvers*, from a sequentially secure PSR system. When all *secondaries* get the exact same pair of keys we prove that non-interactive PSRs remain concurrently secure as well.

We denote by $\{view_A^{SIM}(f(R))\}$ the view which contains $f(R), PK^*$, the random tape of A and the concurrent interaction between SIM and A. We denote by $\{view_A^{real}(R)\}$ the view which contains $f(R), PK$ the random tape of A and the concurrent interaction between the real PSR system and A, where the keys are generated by the setup algorithm of the PSR and the provers are honest *secondaries* in a real PSR system.

Definition 5. *A PSR system is f-Concurrent Zero Knowledge (f-CZK) if for every nonuniform probabilistic polynomial time concurrent adversary A and every $R \subseteq U$ there exists a* concurrent PSR simulator *SIM, such that the two views: $\{view_A^{SIM}(f(R))\}$ and $\{view_A^{real}(R)\}$ are indistinguishable, even for a distinguisher which knows R.*

Note that the way we defined the f-ZK simulator in Section 2.3 the simulation occurs online, meaning there is no rewinding. Rewinding usually raises an obstacle in going from regular ZK to concurrent ZK, so this is a good property to have for the simulator. We prove that a non interactive PSR system (one-round proofs) is always an f-CZK PSR system. On the other hand, we show that for many-round PSR systems this is not necessarily the case: we provide a counter example with more than one round proofs which is not concurrent zero knowledge.

Theorem 2. *If $(Setup, Prove, Verify)$ constitute an f-ZK PSR system with one round proofs then it is also f-*Concurrent *Zero Knowledge.*

Proof. Assume towards contradiction that there exists a concurrent adversary A such that there exists a distinguisher D, that can distinguish between an interaction of A with a real PSR system and an interaction of A with a concurrent PSR simulator. We describe an adversary B which uses A as a subroutine in order to generate two views (one of B interacting with the system and one interacting with the f-ZK simulator) which can be distinguished with a non-negligible advantage. B simply acts as a mediator between the concurrent adversary A and the prover (system/simulator). Every time A issues a new query to some prover, B simply sends the first message of the interaction to the prover and records the response. Notice that although A might be asking for different provers, B only uses the one prover it has access to and as this is only a two message protocol, B

simply records the response to the query. When A asks for the response of that interaction, B sends back the recorded response. When A wishes to terminate the interaction, B terminates the interaction with the prover. At the end of the interaction the view generated by the adversary B isn't in the concurrent setting as in practice B executed the interactions with the provers sequentially. This is not a problem as we can describe a distinguisher D' which uses D to distinguish between interactions with an f-ZK PSR simulator and ones with a real PSR system.

When D' gets a view of the interaction between B and the prover it also gets B's random tape, so D' can run it again with the same random bits (the random tape is included in the view) and rearrange the view it got to look like a concurrent view (i.e. rearrange the order of the messages). Now D' runs D on the newly generated view and outputs its output. If D succeeds in distinguishing between the provers with non-negligible advantage ε, then so does D' as the view of the adversary A interacting with the provers is identical to the view of B interacting with the same provers after D' completed its transformation of the view to look concurrent. Thus we reach a contradiction, which means that non-interactive f-ZK PSR systems are also f-concurrent zero knowledge in the same sense: computational, statistical or perfect ZK. □

Counter example for a many-round PSR: We show that Theorem 2 does not hold when we try to generalize it to many-rounds PSRs. Suppose that we have a one-round proof f-ZK PSR. We modify it by adding two more rounds to its proof. During the setup algorithm the *primary* selects some pseudorandom function F, such that for an adversary (who doesn't know the secret key), the probability of guessing $F(x)$ for a randomly chosen x will be negligible in the security parameter for the PSR. The first round of the interaction will be the *resolver* asking to learn the value $F(x_1)$ for x_1 of its choice (under honest behavior it should be uniformly random). The second round will be the *secondary* sending an element x_2, chosen uniformly at random, to the *resolver* and if the *resolver* returns the correct value $F(x_2)$ then the *secondary* returns a description of R. Otherwise it continues to the original one round proof of the PSR. One can see this is still an f-ZK PSR, as guessing $F(x_2)$ for a randomly chosen x_2 is successful with only negligible probability, even after seeing several values of F. Thus the *resolver* will learn more than it should about R only with negligible probability, making the new PSR secure if the original one was secure.

On the other hand, in a concurrent setting, a malicious *resolver* can simply interact with a *secondary* and when it gets its challenge x_2, stop the interaction and start a new one with a new *secondary*. In the first round, the *resolver* will set $x'_1 = x_2$, i.e. it asks the new *secondary* what is the value of $F(x_2)$; it will then return the answer to the first *secondary*, which should accept it as the correct answer and then it will "spill the beans" and reveal the entire set R, thus violating the f-CZK property (no concurrent simulator can do it for a random set R).

Concurrent Zero-knowledge with independent keys: The reason the above counter example was successful is that the provers were confined by the common key of the PRF they all shared. We claim that in case we have a concurrent execution of the PSR system but where each prover (*secondary*) - verifier (*resolver*) couple receives different and independently chosen keys (that is for each *secondary*-*resolver* the *primary* executes the setup algorithm independently), then the resulting PSR systems are f-CZK[3].

Proof Sketch: the way the concurrent simulator will work is by running the (regular) simulator for each *secondary* independently. We now use a hybrid argument to show that if we are in the described setting and we have an adversary A that can generate two *distinguishable* views for the concurrent setting, then we can construct an adversary B that can generate *distinguishable* views for the sequential setting. If there is a distinguisher D that can distinguish with non-negligible advantage between the two views (generated by A) then it can also distinguish between at least two adjacent hybrids with non-negligible advantage, due to the hybrid argument. This means that there is some index i for which we can construct the adversary B as follows: the first $i-1$ provers will be simulated by B to be a real PSR system *secondaries* (this is done by running the setup algorithm $i-1$ times), the i^{th} prover will be the prover interacting with B (either a simulator or a real *secondary*) and the rest of the provers will be simulated by B using the strategy employed by the (regular) simulator. The two possible views resulting from interacting with this adversary B will be distinguishable with a non-negligible advantage due to the hybrid argument, thus contradicting the assumption that the PSR system is f-ZK. □

Remark 3. In the full version of the paper [34] we claim and give a proof sketch to show that in the Universally Composable security (UC security) framework, introduced by Canetti [10], PSR systems which have non-interactive proofs or use independent keys are also secure in the UC framework.

4 HIBE Based Construction of PSR Systems

In this section we introduce a PSR system based on *Hierarchical Identity Based Encryption* (or HIBE for short). We think of the universe of elements U, as the leaves of a full binary tree. The *primary* can generate an encryption key for any node in the tree, where this encryption key holds the power to prove non-membership for every element in the universe which is a descendant of that node. A proof of non-membership for an element $x \in U$ uses the encryption key of the leaf that corresponds to x, while an encryption key for an internal node can generate the keys of its descendants. Thus if the *primary* generates the

[3] Note that it is critical to use different keys for every couple (*secondary*-*resolver*) running concurrently, otherwise in the scenario described in the counter example, either a malicious *resolver* can communicate with two *secondaries* using the same keys and break the f-CZK property, or two malicious *resolvers* can collide and interact with one *secondary* using the same keys to break the f-CZK property.

encryption key for the root node, it can then generate a set of keys K which contains keys *only* to the elements in $U \backslash R$. In order to do that the *primary* removes the entire path of keys from the root to a leaf $x \in R$ and generates keys to the siblings of each node along that path. One might notice the similarity to revocation schemes, as we "revoke" all keys for the elements in R and as shown by Naor et al. [31], this process results in a forest of $O(|R| \cdot \log \frac{|U|}{|R|})$ full binary trees (See Figure 2 for an example).

In order to generate this set of keys K we will use a HIBE scheme, which is an identity based encryption scheme (i.e. an element's encoding is its identity) with the special property we need: that every key can generate keys to its descendants in the hierarchy tree. For high efficiency we use the HIBE construction of Boneh et al. [6], which we describe in more details in Section 4.4. Agrawal, Boneh and Boyen also offer two HIBE constructions [3,2] based on lattices, which give us also two lattice based assumptions from which we can construct a PSR system. The HIBE construction is *perfect ZK*, in the sense that it doesn't reveal any information about the set R to any adversarial *resolver*, not even its cardinality, while providing perfect simulation.

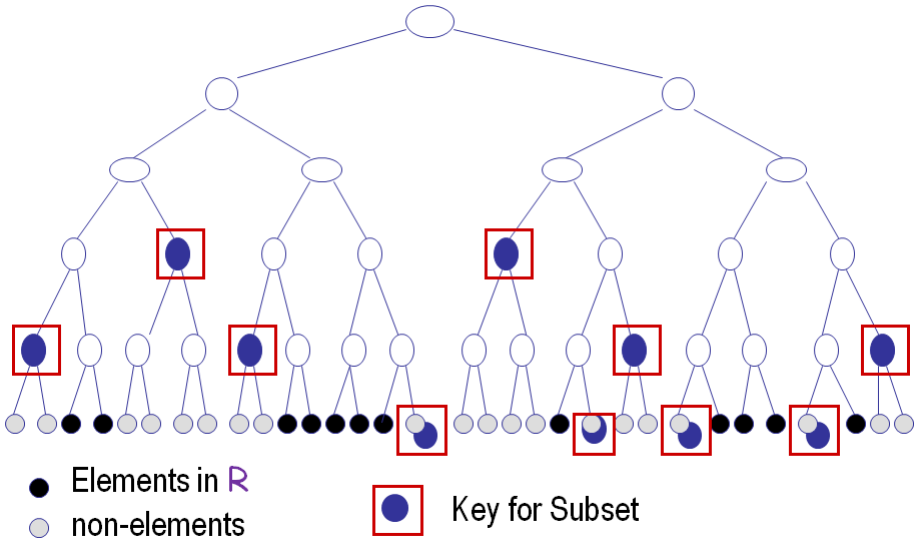

Fig. 2. A full binary tree that represents a set R and its set of keys K.

4.1 HIBE Definition

An IBE (Identity Based Encryption) is a scheme where one can encrypt messages to users using their names/IDs or any other unique identifiers one chooses to use.

A trusted party generates a master public key (also called system parameters sometimes) and a master secret key, where the first is used by users to encrypt messages under any identity they wish, while the latter is used to generate secret keys for all identities in the scheme, which are then distributed to the users (each user gets its own secret key). A user can then use its secret key to decrypt messages intended for him. A HIBE is an hierarchical IBE, which means that identities in the scheme are defined by up to ℓ coordinates and anyone who has a secret key for its identity x, can generate secret keys to any of its descendants, i.e. to any identity with x as its prefix.

We use the following definition for HIBE which is similar to that of Gentry and Silverberg [17]. An **ID-tuple** is a description of a user in the system defined by $(I_1, \ldots, I_t)$ where $t \leq \ell$ and ℓ is the maximum depth of the hierarchy of identities, i.e. the maximal number of coordinates in an identity. In our construction we use binary vectors as the identities.

Definition 6. *A HIBE is defined by five algorithms: Setup, MKeyGen, KeyGen, Encrypt and Decrypt.*

Setup *Gets a security parameter k and the depth of the hierarchy ℓ and generates the master public key MK_P, which should be distributed to all the users in the system and a master secret key MK_S given only to the root user, both corresponding to the HIBE of depth ℓ.*

MKeyGen *Gets the master key MK_S and a target identity $ID = (I_1, \ldots, I_t)$ and generates a private key (from a distribution of valid keys) denoted as SK_{ID}, which user ID can use to decrypt messages intended for him and also to generate properly distributed private keys (i.e. with same distribution, as if it was generated using MK_S) to any of its descendants (any user who has the identity ID as a prefix to its own identity).*

KeyGen *Gets a private key SK_{ID} for identity $ID = (I_1, \ldots, I_t)$ and some descendant of that identity of any level, $ID^* = (I_1, \ldots, I_t, I_{t+1}, \ldots, I_m)$ and generates a private key SK_{ID^*} from its proper distribution. It is critical that for every identity, two different ancestors produce the same distribution on the generation of its private key. Sometimes this algorithm is described only for one level deeper than that of ID, but this can be extended by invoking the algorithm recursively.*

Encrypt *Gets the master public key, a message m and a target identity ID and outputs a ciphertext CT which is an encryption of m intended for ID.*

Decrypt *Gets a private key for identity ID and a ciphertext CT intended for that identity and decrypts it to retrieve the original message m.*

We include the description of the HIBE by Boneh et al. [6] in Section 4.4, which is the most efficient HIBE implementation we could find for our purposes. It uses only a constant number of pairing computations and exponentiations and a logarithmic number (in the size of the universe U) of multiplications in a group, for the algorithms used by the *secondaries* and *resolvers*: Encrypt, Decrypt and KeyGen for leaves in the tree. Not all algorithms are as efficient as those three, but we may allow the *primary* setup to take longer time as it commits to the set R only once.

4.2 HIBE Security

There are four types of security notions for HIBE. We have indistinguishability under chosen plaintext attack and under chosen ciphertext attack, where in the first an adversary can issue queries to different secret keys in the HIBE and in the second it can also issue decryption queries where it can ask to decrypt ciphertexts. For the needs of our construction the weaker notion of security will suffice, i.e. indistinguishability under chosen plaintext attack. We can also talk about the difference between selective and existential security, where in the first an adversary selects a priori the target identity it wishes to be tested on and in the second it can choose the target identity after it issues some queries. Again we only need the weaker notion of security for our construction, i.e. selective security. We use the definitions of security as defined by Boneh et al. [6].

Definition 7. *Indistinguishability under selective identity chosen plaintext attack (IND-sID-CPA). We say that a HIBE system is (t, q, ε) IND-sID-CPA if any t-time adversary A that uses q queries wins the following game with an advantage of at most ε. This is a communication game between an adversary A and a challenger which controls the HIBE system at hand.*

step 1: *A sends a target identity ID^* to the challenger and two equal length messages m_0, m_1 on which it wishes to be tested.*

step 2: *The challenger runs the HIBE's setup algorithm, sends the master public key to the adversary and keeps the master secret key to himself.*

step 3: *A adaptively issues up to q key queries to the challenger, where it asks to know the private key of an identity ID. The challenger responds with the correct keys to all queries. The only restriction is that A didn't issue a key query on identity ID^* or a prefix of it.*

step 4: *The challenger draws a bit at random $b \in \{0, 1\}$, computes $CT = Encrypt(MK_P, ID^*, m_b)$ and sends CT to A.*

step 5: *A issues more queries (where the total number of queries is at most q) where again A cannot issue key queries to prefixes of the identity ID^* or to ID^* itself. When A finishes with the queries it issues a guess $b' \in \{0, 1\}$ and wins the game if $b' = b$.*

Notation. If we have a HIBE which is (t, q, ε) IND-sID-CPA secure, t, q are polynomials and ε is negligible in the scheme's security parameter, then we simply say it is IND-sID-CPA secure.

Remark 4. In a recent paper, Lewko and Waters [27] examine the difficulty in proving full (existential) security for HIBEs. They show that proving full security for a large class of HIBEs results in an exponential degradation (in the depth of the hierarchy) in security. Luckily for us we only need selective chosen plaintext security (the weakest security notion for HIBEs), which most if not all HIBEs achieve, without the exponential degradation.

4.3 PSR from HIBE

Suppose that all possible queries that *resolvers* issue are in the domain $\{0,1\}^\ell$. We can assume that, as we may use a collision resistant hash function h in order to map our domain of queries into a domain with the appropriate ℓ. We will use a HIBE of depth ℓ. As we do in all constructions, for $x \in R$ we will use *consistent* signatures on the element and its value, i.e. a signing algorithm that produces the same signature on the same message. We will use the HIBE scheme to deal with non-membership proofs. In order to prove non-membership in R, the *secondaries* will get as part of the secret key SK, a set of secret HIBE keys K, from which they can generate a secret key corresponding to any $x \notin R$ (the secret key is $SK_{h(x)}$) and prove its possession by decrypting random challenges encrypted by the *resolvers* under the queried element's identity $h(x)$ (alternatively the key may be given to the *resolvers* who should verify its correctness).

We do not want the *secondary* to be able to prove the non membership of an actual member $x \in R$, so we make sure it cannot obtain the secret keys to any element in R. Thus *secondaries* will not be able to prove false statements with overwhelming probability, as in order to prove false statements the *secondary* will have to either forge signatures or decrypt a message it doesn't have the private key for.

In order to give *secondaries* the correct set of private keys, consider the full binary tree of depth ℓ. The *primary* removes all nodes which are in R or are ancestors/prefixes of elements in R. All the remaining nodes in the tree (both internal and leaves) comprise a forest of full binary trees of different depths. The *primary* then generates the secret key to all the roots of the binary trees in the forest and distributes it to the *secondaries*. Now, the union of all those keys, denoted as K, can generate all keys corresponding to leaves that are not members of R. As mentioned before, the number of trees in the forest can be shown to be $O(r \log \frac{|U|}{r})$ [31].

We now describe the PSR construction that uses a HIBE which is required to be *only* IND-sID-CPA secure (see Definition 7 for details) and an existentially unforgeable signature scheme.

$Setup(R, V, 1^k)$**:** Use the setup algorithm for the signature scheme in order to obtain the keys (PK_{sig}, SK_{sig}, h) where h is a collision resistant hash function that maps elements from U to $\{0,1\}^\ell$. Use the setup algorithm for the HIBE scheme and obtain the master public key MK_P and the master secret key MK_S for a HIBE of depth ℓ. Set the public key to be $PK = (PK_{sig}, MK_P, h)$.
Now generate the forest of full binary trees, as specified above, by removing all the nodes in the full binary tree of depth ℓ, which are prefixes of $h(x_i)$ for every $x_i \in R$. For every root t_j in that forest, generate its corresponding secret key k_j (using the MKeyGen algorithm) and set $K = \{(t_j, k_j)\}$. Now sign every element $x_i \in R$ with its value: $s_i = (Sign_{SK_{sig}}(x_i, v_i), (x_i, v_i))$ and set the secret key to be $SK = (K, \{s_i\}_{i=1}^r)$.

$Verify(x, PK)$**:** Gets an element $x \in U$ and the public key and initiates an interactive protocol with a *secondary*. It draws uniformly at random a mes-

sage m from the message domain of the HIBE scheme and encrypts it under the public key of $h(x)$: $CT = Encrypt(m, h(x), MK_P)$. It send (CT, x) to a *secondary*. If it gets in return back m, it returns 1 and "$x \notin R$"; if it gets in return a signature s and a pair (x, v) where it verifies correctly that s is a valid signature on (x, v) then it accepts that $x \in R$ and its value is v and returns 1. Otherwise it returns 0.

Prove(x, PK, SK)**:** Gets the public and private keys and also (CT, x) from a *resolver*. If there exists a signature s_i for which $x_i = x$, then it returns s_i. Otherwise the secret key SK contains, in its HIBE set of keys K, a key for a prefix of $h(x)$. The *secondary* generates the secret key for $h(x)$ (using the HIBE KeyGen algorithm), decrypts CT under that secret key and returns m to the verifier.

Theorem 3. *The three algorithms described above constitute a (perfect) ZK PSR (i.e. f is the null function and the simulation is perfect).*

Proof. In order to prove the above scheme constitutes a PSR system we need to prove it fulfills the three properties required from a PSR system: completeness, soundness and zero-knowledge.

Perfect Completeness. For all $R \subseteq U$, for all V and for all $x \in U$ we need to show that after obtaining the keys (PK, SK) from the setup algorithm, it always holds that an honest *secondary* manages to convince an honest *resolver* of the true statement regarding the queried element x. For every element $x_i \in R$ the *primary* precomputed $s_i = (Sign_{SK_{sig}}(x_i, v_i), (x_i, v_i))$ which is part of the secret key and thus the *secondary* will always succeed in proving membership statements. As for statements of the type $x \notin R$, using the set of HIBE keys K given to the *secondaries*, they can derive a secret key for every $x \in U \backslash R$ (actually a key for every such $h(x)$). Using that key $SK_{h(x)}$, *secondaries* can always decrypt a random challenge issued by *resolvers* and thus will always manage to prove statements of non-membership.

Soundness. Assume for contradiction that there exits some polynomial time adversary that using (PK, SK) can provide for some $x \notin R$ a proof that $x \in R$ with non-negligible probability. This means it can forge a signature with non-negligible probability for that x and some value v, violating the unforgeability assumption on the underlying signature scheme. The same holds if an adversary is trying to prove for some $x \in R$ with value v a different value $v' \neq v$, i.e. due to the existential unforgeability of the signature scheme proving a false value for $x \in R$ is infeasible as well.

If we assume to have such an adversary A that can provide for some $x \in R$ a proof that $x \notin R$ with non-negligible probability ε, then we can use A to construct an adversary B that wins the IND-sID-CPA security game (Definition 7) with a non-negligible advantage $\frac{\varepsilon}{2}$. If A can cheat with probability ε for the set $R \subseteq U$ and some $x \in R$ then the adversary B (trying to win the IND-sID-CPA security game) will first select $h(x)$ as its target identity (h will be chosen by him as well), choose two random messages as the challenge messages $\{m_0, m_1\}$ and get the HIBE master public key, MK_P. Then B runs the setup algorithm for

the PSR over U and R while using MK_P as its master public key for the HIBE in the PSR and will use the key queries in the security game to generate the set of HIBE keys K. Note that as $x \in R$ all the key queries will be for non-prefixes of $h(x)$ as K doesn't contain any ancestors of $h(R) = \{h(x_i)|x_i \in R\}$.

Thus B will generate a valid pair of keys (PK, SK) for a PSR and hand them to the adversarial *secondary* A. B will now send the random challenge it got form the IND-sID-CPA security game (an encryption under $h(x)$ of m_0 or m_1) to A which will try to decrypt the ciphertext. A succeeds in decrypting the challenge with probability ε and if the decryption A offers matches one of the two original challenge messages (m_0, m_1) then B chooses this message and else it guesses uniformly at random. Thus B wins the IND-sID-CPA security game with an advantage of about $\frac{\varepsilon}{2}$ [4]. Thus violating the security assumption made on the HIBE scheme being used.

We also note that it is infeasible for an adversary to find an element on which it can provide a false proof. As the adversary gets both keys we can assume it knows R. The adversary cannot find an element $x \notin R$ and provide a false proof with non-negligible probability as this again violates the unforgeability of the signature scheme. Regarding $x \in R$ as we know that the HIBE is *selectively* secure then we know that if the target identity is chosen in advance, then any polynomial time adversary has at most a negligible advantage ε in distinguishing between the two target messages, which makes its probability of decrypting the target ciphertext at most 2ε (by the reduction shown above). So as this time there are $|R| = r$ target identities, any adversary has at most a probability of $2\varepsilon \cdot r$ (still negligible as r is polynomial) to decrypt a random challenge under one of the identities of $h(R)$, thus it is also infeasible to find $x \in R$ for which a *secondary* can cheat on.

Perfect ZK. In order to show that this PSR is indeed zero knowledge we need to show a suitable simulator SIM which can fool any adversary into believing it is a real PSR system. SIM simply chooses the function h as the *primary* does, runs the setup algorithm for the HIBE to obtain (MK_P, MK_S) and the setup algorithm for the signature scheme to obtain (Pk_{sig}, SK_{sig}). SIM then sets the fake public key to be $PK^* = (MK_P, PK_{sig}, h)$ and the fake secret key to be $SK_{SIM} = (SK_{sig}, MK_S)$. Note that the fake public key is generated the exact same way the original public key is generated and the fake secret key has the master secret key for the HIBE instead of the subset of the keys (K) and the secret key for the signature scheme instead of the signatures on the elements of R and their values $(\{s_i\}_{i=1}^{r})$. When SIM is queried on an element $x \in U$, it queries its oracle to R on x. If $x \in R$ and its value is v_x it returns $s = (Sign_{SK_{sig}}(x, v_x), (x, v_x))$. If $x \notin R$ then SIM gets (CT, x) and it can generate the secret key for $h(x)$ using the master secret key MK_S, decrypt the challenge and return it to the adversary.

[4] There is a probability that A decrypts CT to a wrong message that happens to be m_{1-b} while m_b was chosen as the challenge. But, as $\{m_0, m_1\}$ are chosen uniformly at random and are not known at all to A this probability is negligible.

We claim that the two views generated by the simulator and a real PSR system are not only indistinguishable but identically distributed, thus making this construction **perfect** zero-knowledge. The public keys are generated by the same algorithm. The signatures (proofs regarding $x \in R$) are generated online instead of during the setup algorithm as the *primary* does, but yield the same distribution over the signatures, due their consistency. Proofs for elements $x \notin R$ are also identical as both the simulator and a PSR system decrypt successfully the random challenges on elements outside of R with probability 1 and simply return it. This concludes the proof that this PSR system is perfect ZK. □

Remark 5. Note that we can also use two variants of HIBEs, one where *secondaries* deliver the queried element's decryption key to the *resolver* (and by that make it verify the key's correctness by itself) and one where we use signatures instead of encryption, i.e. *secondaries* produce signatures over the queried element with its corresponding secret key.

4.4 HIBE Construction by Boneh, Boyen and Goh

We describe the construction by Boneh et al. [6] as it is the most efficient HIBE implementation for our needs. Its greatest virtue, with respect to our construction, is the fact that generating secret keys for nodes get more efficient the deeper the node is in the hierarchy. Thus generating keys for leaves is very efficient, which is critical for us, since this is done online by the *secondaries* generating non-membership proofs. Let $\mathbb{G}$ be a bilinear group of prime order p and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ be an admissible bilinear map (i.e. its bilinear- $\forall g_1, g_2 \in \mathbb{G}$ it holds that $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$, non-degenerate - $e(g, g) \neq 1$ and efficiently computable). We choose arbitrarily how to map J_0, J_1 to $\mathbb{Z}_p^*$, since the original HIBE can handle identities of the type $ID \in (\mathbb{Z}_p^*)^\ell$ (or shorter), while we only require binary identities of length at most ℓ. This means that for some node in level k of the tree, $u = x_1 \dots x_k$ where $x_i \in \{0, 1\}$ has identity $I_u = (J_{x_1}, \dots, J_{x_k}) = (I_1, \dots, I_k)$, which will be also its public key. We also assume that the messages to be encrypted are elements in $\mathbb{G}_1$. We choose ℓ, the depth of the hierarchy, to be $\lceil \log |U| \rceil$, in order for the leaves of the full binary tree of depth ℓ to represent the elements in the universe.

The HIBE system works as follows:

- $Setup(1^k, 1^\ell)$: Gets k the security parameter and ℓ the depth of the hierarchy. To generate the public master key for the HIBE of maximum depth ℓ, draw uniformly at random: $g \in \mathbb{G}$, $\alpha \in \mathbb{Z}_p^*$, set $g_1 = g^\alpha$ and pick some more random elements $g_2, g_3, h_1, \dots, h_\ell \in \mathbb{G}$. Next compute $Aux = (h_1^{J_0}, h_1^{J_1}, \dots, h_\ell^{J_0}, h_\ell^{J_1})$ and define the master secret key to be $MK_S = g_2^\alpha$ and the public master key to be: $MK_P = (g, g_1, g_2, g_3, h_1, \dots, h_\ell, Aux)$.
- $MKeyGen(MK_S, ID)$: To generate a private key for $ID = (I_1, \dots, I_k) \in (\mathbb{Z}_p^*)^k$ pick uniformly at random $r \in \mathbb{Z}_p$ and output:

$$SK_{ID} = (g_2^\alpha \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^r, g^r, h_{k+1}^r, \dots, h_\ell^r) \in \mathbb{G}^{\ell-k+2}$$

Note that the deeper the node the smaller the private key.

- $KeyGen(SK_{ID}, ID^*)$: For $ID^* = (I_1, \ldots, I_m) \in (\mathbb{Z}_p^*)^m$ and a private key of its ancestor $ID = (I_1, \ldots, I_k)$ $(m > k)$ do the following in order to generate a properly distributed key:
  If $SK_{ID} = (g_2^{\alpha} \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^{r'}, g^{r'}, h_{k+1}^{r'}, \ldots, h_{\ell}^{r'}) = (a_0, a_1, b_{k+1}, \ldots, b_{\ell})$ then choose uniformly at random $t \in \mathbb{Z}_p$ and output: $SK_{ID^*} =$

  $$(a_0 \cdot b_{k+1}^{I_{k+1}} \cdots b_m^{I_m} (h_1^{I_1} \cdots h_m^{I_m} \cdot g_3)^t, a_1 \cdot g^t, b_{m+1} \cdot h_{m+1}^t, \ldots, b_{\ell} \cdot h_{\ell}^t) \in \mathbb{G}^{\ell - m + 2}.$$

  This can be computed using $4 + (\ell - m)$ exponentiations and $O(\ell)$ multiplications by utilizing Aux. This private key is a properly distributed key for $ID^* = (I_1, \ldots, I_m)$ with $r = r' + t \in \mathbb{Z}_p$. Note that the deeper the node – the shorter the key, thus computing a secret key for a leaf is very efficient. If ID^* is a leaf $(m = \ell)$ we get:

  $$SK_{ID^*} = (a_0 \cdot b_{k+1}^{I_{k+1}} \cdots b_{\ell}^{I_{\ell}} (h_1^{I_1} \cdots h_{\ell}^{I_{\ell}} \cdot g_3)^t, a_1 \cdot g^t) \in \mathbb{G}^2.$$

  Computing secret keys for the leaves takes only 4 exponentiations and $O(\ell)$ multiplications, since by utilizing Aux, the *secondary* multiplies all the b_i's where $I_i = J_1$ and then raises them to the power of J_1 and similarly for J_0; exponentiations of $h_i^{J_j}$ are already calculated and included in Aux.
- $Encrypt(MK_P, ID, m)$: To encrypt a message $m \in \mathbb{G}_1$ under the public key $ID = (I_1, \ldots, I_k)$ draw uniformly at random $s \in \mathbb{Z}_p$ and output:

  $$CT = (e(g_1, g_2)^s \cdot m, g^s, (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^s) \in \mathbb{G}_1 \times \mathbb{G}^2$$

  Which takes 1 pairing computation, 3 exponentiations and $O(\ell)$ multiplications (we can also add $e(g_1, g_2)$ to MK_P in order to avoid computing pairings in the encryption).
- $Decrypt(SK_{ID}, CT)$: Consider a ciphertext $CT = (A, B, C)$ encrypted for $ID = (I_1, \ldots, I_k)$ where the private key is $SK_{ID} = (a_0, a_1, b_{k+1}, \ldots, b_{\ell})$. Output:

  $$A \cdot \frac{e(a_1, C)}{e(B, a_0)} = e(g_1, g_2)^s \cdot m \cdot \frac{e(g^r, (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^s)}{e(g^s, g_2^{\alpha} \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^r)} =$$

  $$= e(g_1, g_2)^s \cdot m \cdot \frac{1}{e(g, g_2)^{s\alpha}} = m$$

  Which takes only two pairing computations and one multiplication.

This HIBE achieves selective-ID security for both chosen plaintext and chosen ciphertext attacks (IND-sID-CPA and IND-sID-CCA respectively) under the ℓ-weak decisional Bilinear Diffie-Hellman Inversion assumption (ℓ-wBDHI, see definition in [6]) in the standard model and is fully secure in the random oracle model, where ℓ is the number of levels of the hierarchy.

Performance. As for the performance of the resulting PSR, the setup algorithm's running time is dominated by the generation of the set of private keys K

which is of size $O(r \log \frac{|U|}{r})$. In order to provide proofs of non-membership, the *secondaries* have to decrypt a message intended for an identity of depth ℓ, for which they have to first generate a proper key. This takes 4 exponentiations and $O(\ell)$ multiplications. The *secondaries* then decrypt the message, which takes 2 pairing computations and one multiplication. For a *resolver* to issue a query for an element it has to encrypt one message which takes 3 exponentiations and $O(\ell)$ multiplications (we avoid the pairing computation in the encryption by adding $e(g_1, g_2)$ to MK_P).

So in total a *secondary* has to do at most 2 pairing computations, 4 exponentiations and $O(\ell)$ multiplications, while a *resolver* has to do only 3 exponentiations and $O(\ell)$ multiplications. As mentioned before, we can also have a variant of the protocol where the *resolvers* receive the secret key itself (and have them encrypt and decrypt random challenges by themselves). This moves the computational load of 2 pairing computations to the *resolvers*. The *primary* has to work harder as the setup algorithm is more costly, but that is understandable as the *primary* has to set up the system only once.

5 PSR system Constructions

In the full version of this paper [34] we present two additional strategies for constructing PSR systems and another construction which follows the lines of the HIBE construction but uses one-time signatures. We describe them here informally, where the full version contains a more comprehensive and formal treatment of these constructions.

5.1 Using One-time Signatures

One-time signatures are signatures with a very weak security/unforgeability requirement, where an adversary who witnesses at most one signature of its choice cannot forge a signature, which will be verified successfully. We utilize the same strategy we used for the HIBE construction (Section 4), with the difference of using one-time signatures to produce a chain of signatures from the root of a binary tree to the leaf corresponding to the queried element (again where a *secondary* cannot generate this proof for elements in the set R). The chain of signatures consists of public keys corresponding to the nodes along the path, signed using the secret key of their parents. In the full version of this paper [34] we prove this construction is a non-interactive PSR system and has perfect ZK.

Now as we can construct both types of signatures (one-time and regular) from universally one way hash functions (UOWHF) [33], we can conclude that the existence of UOWHFs implies the existence of PSR systems with perfect ZK. UOWHFs in turn can be constructed from one-way functions [39]. PSR systems imply identification schemes, as shown in our companion paper [19], which in turn imply the existence of one-way functions, as shown by Impagliazzo and Luby [24] (see also [23]).

Thus the point of this construction is not efficiency, but to use this black box constructions to prove the following corollary:

Corollary 1. *Single round PSR systems exist if and only if one-way functions exist. If many rounds PSR systems exist then a single round PSR system exists.*

This also gives us a separation result from ZKS [29], since Chase et al. [12] proved that interactive ZKS and collision resistant hash functions (CRH) are existentially equivalent and Simon [43] showed a separation result, which states that no CRH can be constructed from one-way functions (or even permutations) in a black box manner. Thus we get the following corollary:

Corollary 2. *One cannot construct ZKS (and even interactive ZKS) in a black box manner from PSR systems (interactive or not).*

5.2 Using Cuckoo Hashing with a Stash

We now discuss an instantiation of the second approach for constructing PSRs mentioned in the introduction, imitating an oblivious search for the element, where the locations examined are determined by the element searched and some hash functions. The point is that the *secondary* needs to show that the searched element is in none of the probed locations.

Cuckoo Hashing is a scheme first introduced by Pagh and Rodler [36]. If we have a set $|R| = r$ for which we want to prove (non) membership, we use two tables T_1 and T_2 of size $(1+\varepsilon)r$ (where ε is constant) and two hash functions (F_1, F_2), which map elements in the universe into those two tables. Every element $x \in R$ is placed in either location $F_1(x)$ in table T_1 or location $F_2(x)$ in table T_2. This off course may fail for the choice of some functions (F_1, F_2) (with probability $O(\frac{1}{r})$), thus we also use a stash, to store elements we could not place in the cuckoo hash tables due to collisions. Kirsch, Mitzenmacher and Wieder [26] show that the probability that the stash is larger than s is bounded by $O(r^{-s})$. This helps us bound the amount of information that leaks on the set R, by the choice of the functions (F_1, F_2). In order to prove $x \notin R$ we need to show that x was not placed in the stash and not in either of the two possible locations in the cuckoo hash tables.

In order to prove non-membership in the tables we use commitment schemes (see [20] for definitions) with inequality proofs, where we require the commitments to be: *hiding*, so that commitments to two different values are identically distributed, and *binding*, so that even the commiter cannot open a commitment to a value, different than the committed value. We also want the proofs of inequality to be complete (honest execution results in correct conclusions), sound (commiter can't cheat) and have indistinguishability between two proofs of inequality, i.e. proving the inequality of x to two commitments to elements different than x is indistinguishable. In order to prove an element was not placed in the stash we use a scheme for proving non-membership in a fixed set, from which we require the exact same conditions as we require from commitments, with the difference that we need to commit to a set of elements instead of a single element.

We chose to use Pedersen commitments [37] with ZK proofs of inequality. The inequality proofs use ZK proofs of equality for Pedersen commitments (based on the adaptation of Schnorr's identification protocol [41]) and the ZK proofs of inequality for discrete logarithms, suggested by Camenisch and Shoup [9]. In order to construct a scheme to prove non-membership in a fixed set (our stash S), we use a generalization of the Feige-Fiat-Shamir identification protocol [15] combined with the set lower bound technique of Goldwasser and Sipser [21], to allow *secondaries* to prove they know a *large* fraction of the secrets (corresponding to the queried element) as opposed to knowing none of them (when the queried element is placed in the stash). Every element in the universe is mapped to n challenges and the *primary* distributes the corresponding secret to every challenge that doesn't correspond to an element in the stash S. This way we get that for every $x \in S$ the *secondaries* know none of the secrets, but they know a large fraction of the secrets for every element $x \notin S$.

All and all we get an interactive PSR system which leaks the cardinality of the set R and is quite efficient. The denial-of-existence mechanism we described requires a constant number of exponentiations for both parties (9 for the *secondary* and 8 for the *resolver*) in order to prove inequalities for Pedersen's commitments and at most $n = \log |U|$ modular multiplications and a Gaussian elimination process (for a matrix of size $\frac{n}{4} \times \frac{n}{3}$), for the fixed set non-membership proof system, suggested to implement the stash. Its great advantage is that it uses very conservative and well studied assumptions: factoring (for the Feige-Fiat-Shamir protocol) and discrete logarithm (for the Pedersen commitments).

5.3 Using Verifiable Random Looking Functions

In the full version of this paper [34] we show a few constructions for PSR systems based on variants of *Verifiable Random/Unpredictable Functions* [30] (VRF and VUF for short), a construction that uses *Pseudorandom Functions with interactive ZK proofs* and discuss constructions in the random oracle model. All these constructions employ the same strategy which uses functions that map elements in the universe to some large domain $\{0,1\}^m$, where a *secondary*, holding a secret key, can prove to a *resolver*, holding a public key, the value of $F(x)$. Another important property we require from our functions is to appear random, in the sense that an adversary (without knowledge of the secret key) who knows the set R, cannot distinguish between the set of values $F(R) = \{F(x)|x \in R\}$ and a set of random values $\{r_i | i \in [r] : r_i \overset{R}{\in} \{0,1\}^m\}$, even after a series of queries to the function (which do not include queries to elements in R)[5].

The PSR system itself does the following: the *primary* computes the values of the function over the set R and arranges them in lexicographical order $y_1, \ldots, y_r$. Next it signs all the couples of adjacent values and gives the signatures to the *secondaries*: $Sign(y_i, y_{i+1})$ (adding an opening and a closing value 0^m and 1^m). Now in order to prove non-membership for an element $x \notin R$ the *secondary* simply

[5] Both VRFs and PRFs achieve this property naturally by their definitions, while we need to modify VUFs a bit in order to get this property.

computes $F(x)$ and finds and index i for which $y_i < F(x) < y_{i+1}$, proves to the *resolver* that it computed $F(x)$ honestly and sends the signature $Sign(y_i, y_{i+1})$. The *resolver* is convinced after verifying that $F(x)$ was computed correctly and that its value is truly in between two values of the set R, precomputed by the *primary* (i.e. it verifies that $y_i < F(x) < y_{i+1}$).

This construction leaks the size of the set R and can be instantiated using different implementations for the function F, thus resulting in different efficiency and PSR systems which are based on different cryptographic assumptions. The VRF and VUF constructions are non-interactive and can be implemented using the constructions of [22,8,1,25] (VRF) and [16,25] (VUF) in the standard model. As a PRF with interactive ZK proofs we suggest the Naor-Reongold PRF [32]. In the random oracle model we can get very efficient implementations by using functions comprised of the BLS signature scheme [7] and random oracles or the NSEC5 construction suggested in our companion paper [19].

6 Conclusions and Future Directions

We introduced PSR systems and presented three general strategies for constructing them, with different implementations for the underlying primitives. Our focus in this paper was on trying to find efficient constructions, based on solid cryptographic assumptions. A construction can be measured by a few standards: efficiency, the underlying cryptographic assumptions and the ZK requirement (for which f does the f-ZK requirement hold and whether it is computational, statistical or perfect ZK). There is no clear overall winner that dominates in all criteria.

If the (null f) ZK property is critical (e.g. in case the *primary* does not want to reveal the size of the set), then the HIBE construction (Section 4) and the signature based PSR (Section 5.1) both achieve perfect f-ZK, where f is the null function. Both schemes are one-round PSRs and hence they are also secure in a concurrent setting (as proved in Theorem 2). The rest of the constructions reveal the size of the set R and do not achieve perfect ZK. The HIBE construction by Boneh et al. is efficient (Section 4.4), as *secondaries* and *resolvers* use only $O(\log |U|)$ group multiplications and a constant number of pairing computations and modular exponentiations for their computations. It is based on the $O(\log |U|)$-weak decisional Bilinear Diffie-Hellman Inversion assumption[6] (ℓ-wBDHI, see definition in [6]). The downside for this scheme is the computational load on the *primary*, which has to compute keys for $O(|R| \log \frac{|U|}{|R|})$ nodes, which may result in superlinear time for generating the scheme's keys, but at least it is only executed once.

[6] Boneh et al. prove this assumption holds in the generic group model [42]. This means that using generic algorithms (ones that don't exploit any special properties of the group elements' encodings), one cannot construct a polynomial time algorithm to break the assumption, which is an encouraging result towards using this assumption.

Our cuckoo hash based PSR construction (Section 5.2) offers both an appealing technique and an efficient implementation, based on very solid and well studied cryptographic assumptions: factoring and the discrete logarithm (defined in the original papers [15] and [37] respectively). If the security of the PSR is the most important thing for its users (e.g. a database containing top secret information), it makes sense to use the cuckoo hashing construction as it is based on two very well studied assumptions and has the statistical ZK property, which gives us everlasting privacy (see Remark 2). This technique's efficiency depends on the implementations of the commitment scheme and the fixed set non-membership, which using the implementations we suggest (see full version of the paper for exact details [34]) results in the *resolvers* and *secondaries* doing a constant number of modular exponentiations and $O(\log |U|)$ modular multiplications, which is about as efficient as the HIBE construction asymptotically.

Our PSR based on random looking functions (Section 5.3) reveals the size of the set R, but has the potential of being very efficient if we can construct a VRF/VUF which is both efficient and secure. We would like to use such a function which is secure for large domains but can be evaluated and verified with, say, a constant number of operations ([13] is that efficient but lacks security), as *secondaries* only have to evaluate the function on the queried element and generate its proof, while the *resolvers* verify the value and one signature. We note that the four secure VRFs [22,8,1,25] are not a lot less efficient than the HIBE construction. If we can implement an efficient division intractable hashing family then we can use the GHR signature scheme [16] to implement the random looking function, which is highly efficient, as computing and verifying each value requires one hash computation and one modular exponentiation. These constructions also have the added advantage of being non-interactive, which also makes them concurrently secure.

If one is willing to live with random oracles, then this technique yields very efficient PSR systems. Both the BLS signature scheme [7] based PSR and the NSEC5 construction described in our companion paper [19] are very efficient, while the first relies on a gap Diffie-Hellman group (see definition in the original paper [7]) and the latter on the RSA hardness assumption (see definition in [19]).

The implementations proposed are fairly efficient, but undoubtedly it is possible to optimize them or come up with other ones. In terms of readiness to deployment, i.e. whether the implementations are mature, then probably HIBE is the best bet unless one is willing to trust random oracles in which case both the BLS and NSEC5 schemes are good.

7 Acknowledgments

We thank our co-authors from [19], Sharon Goldberg, Dimitrios Papadopoulos, Leonid Reyzin and Sachin Vasant for many helpful discussions and Yevgeniy Dodis for suggesting the question of whether single-round PSRs can be based on one-way functions. We thank Pavel Hubáček for carefully reading the paper.

References

1. Abdalla, M., Catalano, D., Fiore, D.: Verifiable random functions: Relations to identity-based key encapsulation and new constructions. J. Cryptology 27(3), 544–593 (2014)
2. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: EUROCRYPT 2010. pp. 553–572. Springer (2010)
3. Agrawal, S., Boneh, D., Boyen, X.: Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In: CRYPTO 2010. pp. 98–115. Springer (2010)
4. Bau, J., Mitchell, J.C.: A security evaluation of DNSSEC with NSEC3. In: NDSS 2010. The Internet Society (2010)
5. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In: EUROCRYPT '93. pp. 274–285. Springer (1993)
6. Boneh, D., Boyen, X., Goh, E.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT 2005. pp. 440–456. Springer (2005)
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptology 17(4), 297–319 (2004)
8. Boneh, D., Montgomery, H.W., Raghunathan, A.: Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In: ACM CCS 2010. pp. 131–140. ACM (2010)
9. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: CRYPTO 2003. pp. 126–144. Springer (2003)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. IACR Cryptology ePrint Archive 2000, 67 (2000)
11. Catalano, D., Fiore, D., Messina, M.: Zero-knowledge sets with short proofs. In: EUROCRYPT 2008. pp. 433–450. Springer (2008)
12. Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial commitments with applications to zero-knowledge sets. In: EUROCRYPT 2005. pp. 422–439. Springer (2005)
13. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: PKC 2005. pp. 416–431. Springer (2005)
14. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: ACM 1998. pp. 409–418. ACM (1998)
15. Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. J. Cryptology 1(2), 77–94 (1988)
16. Gennaro, R., Halevi, S., Rabin, T.: Secure hash-and-sign signatures without the random oracle. In: EUROCRYPT '99. pp. 123–139. Springer (1999)
17. Gentry, C., Silverberg, A.: Hierarchical id-based cryptography. In: ASIACRYPT 2002. pp. 548–566. Springer (2002)
18. Ghosh, E., Ohrimenko, O., Tamassia, R.: Verifiable member and order queries on a list in zero-knowledge. IACR Cryptology ePrint Archive 2014, 632 (2014)
19. Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: NSEC5: provably preventing DNSSEC zone enumeration. IACR Cryptology ePrint Archive 2014, 582 (2014)
20. Goldreich, O.: The Foundations of Cryptography - Volume 1, Basic Techniques. Cambridge University Press (2001)
21. Goldwasser, S., Sipser, M.: Private coins versus public coins in interactive proof systems. In: ACM 1986. pp. 59–68. ACM (1986)

22. Hohenberger, S., Waters, B.: Constructing verifiable random functions with large input spaces. In: EUROCRYPT 2010. pp. 656–672. Springer (2010)
23. Impagliazzo, R.: Pseudo-random generators for cryptography and for randomized algorithms. Ph.D. thesis, University of California, Berkeley (1990)
24. Impagliazzo, R., Luby, M.: One-way functions are essential for complexity based cryptography (extended abstract). In: FOCS '89. pp. 230–235. IEEE Computer Society (1989)
25. Jager, T.: Verifiable random functions from weaker assumptions. IACR Cryptology ePrint Archive 2014, 799 (2014)
26. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. SIAM J. Comput. 39(4), 1543–1561 (2009)
27. Lewko, A.B., Waters, B.: Why proving HIBE systems secure is difficult. In: EUROCRYPT 2014. pp. 58–76. Springer (2014)
28. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: TCC 2010. pp. 499–517. Springer (2010)
29. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: FOCS 2003. pp. 80–91. IEEE Computer Society (2003)
30. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: FOCS '99. pp. 120–130. IEEE Computer Society (1999)
31. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: CRYPTO 2001. pp. 41–62. Springer (2001)
32. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: FOCS '97. pp. 458–467. IEEE Computer Society (1997)
33. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: ACM 1989. pp. 33–43. ACM (1989)
34. Naor, M., Ziv, A.: Primary-secondary-resolver membership proof systems. IACR Cryptology ePrint Archive 2014, 905 (2014)
35. Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. IACR Cryptology ePrint Archive 2004, 170 (2004)
36. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: Algorithms - ESA 2001. pp. 121–133. Springer (2001)
37. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '91. pp. 129–140. Springer (1991)
38. Prabhakaran, M., Xue, R.: Statistically hiding sets. In: Topics in Cryptology - CT-RSA 2009. pp. 100–116. Springer (2009)
39. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: ACM 1990. pp. 387–394. ACM (1990)
40. Rosen, A.: Concurrent Zero-Knowledge - With Additional Background by Oded Goldreich. Information Security and Cryptography, Springer (2006)
41. Schnorr, C.: Efficient identification and signatures for smart cards. In: CRYPTO '89. pp. 239–252. Springer (1989)
42. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: EUROCRYPT '97. pp. 256–266. Springer (1997)
43. Simon, D.R.: Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In: EUROCRYPT '98. pp. 334–345. Springer (1998)