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.

We provide three different strategies to construct a denial of existence mechanism. The first uses a set of cryptographic keys for all elements of the universe which are not members, which we implement using hierarchical identity based encryption and a tree based signature scheme. The second construction uses cuckoo hashing with a stash, where in order to prove non-membership, a secondary must prove that a search for it will fail, i.e. that it is not in the tables or the stash of the cuckoo hashing scheme. The third uses a verifiable "random looking" function which the primary evaluates over the set of members, then signs the values lexicographically and *secondaries* then use those signatures to prove to resolvers that the value of the non-member was not signed by the *primary*. We implement this function using a weaker variant of verifiable random/unpredictable functions and pseudorandom functions with interactive zero knowledge proofs.

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.

Table of Contents

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 [7]).

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 [36] 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 signatures (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 [36] 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 pseudo-random 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 [64] 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 [49] 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 [65] 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 [31]. 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) [55], the Naor-Reingold PRF [59] with zero knowledge interactive proofs, the GHR signature scheme [33] and a random oracle construction which uses the famous BLS signature scheme [18]. The scheme NSEC5 presented in [36] (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* [54], 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 introduce 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. In Section 6 we introduce the cuckoo hashing with a stash based PSR construction. In Section 7 we introduce two general constructions of PSR systems which use a weaker variant of verifiable random/unpredictable functions and analyze different implementations of those primitives. We also introduce a PRF based construction and a random oracle construction. In Section 9 we present concluding remarks. Section 11 is the Appendix and provides standard definitions, tools and assumptions.

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 [54] (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 we prove in Section 5). 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. [25] 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. [24] 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 [52] 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 [54,25] or with arity q as in [24,52]), 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* [67] (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 [11], 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 [63] 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 [35] 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 Section 8.

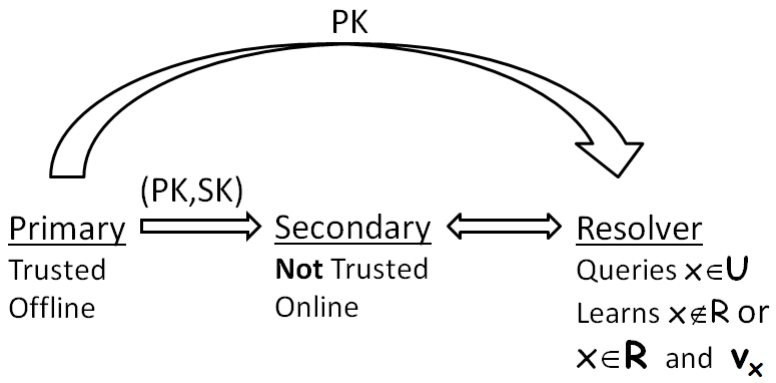

Fig. 1. Illustration of a PSR system.

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 $\forall x \in U$,*

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

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*

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

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), while the rest of the constructions leak the size of the set R (see Sections 6 and 7). 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:

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

and for the simulator the first step is:

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

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 $\mu(k)$:*

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 respectively) achieve perfect ZK, the cuckoo hashing construction (Section 6) achieves statistical ZK, while the last construction (Section 7) achieves computational ZK.

In our companion paper [36], 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

[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.

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 [30] 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 [69] 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 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.

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

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. □

3.1 On Achieving Universally Composable Security

Canetti [22] introduced Universally Composable security (UC security) as a framework for analyzing the security of cryptographic protocols and their composition. In this framework, secure protocols are proven to preserve their security in any context, such as protocol composition and an adversarial network, working concurrently. We claim that PSR systems achieve those demanding security requirements, at least in the cases where we proved it can achieve concurrent security, i.e. when the PSR uses non-interactive proofs or it generates independent keys for each *secondary* and *resolver* in the system.

Without going into too much detail of this framework, in order to qualify as a UC secure protocol, one has to prove that it is infeasible to distinguish, in a concurrently adversarial environment, between a real execution of the protocol and an execution of the "ideal process" of that protocol. In the first scenario we have an environment E where the protocol in question is being executed, where an adversary A has control over a subset of the parties and the scheduling of all messages, while in the second scenario this protocol is replaced with its ideal process, which can be thought of as a trusted party which gets the inputs form the relevant parties and distributes the honestly computed outputs, where using a simulator to replace adversary A (with oracle access to that adversary if needed), one can generate an indistinguishable view from the first scenario. The major difference between concurrent security and universally composable security is the environment which is taken into account. In concurrent security, we require that a simulator is indistinguishable from a real prover (*secondary*) in a **specific** environment where only the discussed protocol is executed concurrently, possibly many times. In the universally composable framework, the simulator should be indistinguishable from the real prover in **any** environment, including ones where different protocols are being executed concurrently and an arbitrary composition of protocols (where one protocol uses the other as a subroutine).

[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.

The ideal process to replace our protocol will be the membership responses to *resolvers*' queries, i.e. getting $x \in U$ as an input and outputting either $x \notin R$ or $x \in R$ and its value v. The observant reader might notice that this functionality is similar to the oracle access we grant our PSR simulator in our f-zk requirement, which is how the simulator in a UC environment is going to use this ideal process to emulate the view of an adversary communicating with an honest PSR system[4]. For both the case of a PSR system with non-interactive proofs and the one with independent keys for each *secondary-resolver* couple, the simulator simply emulates the adversary A (using its oracle access to it) while using the PSR f-ZK simulator, with the ideal process replacing the limited oracle access, to generate proofs for the responses to membership queries asked by *resolvers* in the system.

Like the case in the concurrent setting, non-interactive proofs are not affected by malicious scheduling of messages due to the fact that responses and proofs to queries are only one message, so they can be recorded and sent at a later point in time. Even if *resolvers* try to coordinate a concurrent attack, it is executed sequentially in practice as the *secondary*/simulator generates the response immediately after being queried and saves it until it is asked to send it back to the *resolver*. The use of a different sets of keys for every couple of *resolvers* and *secondaries* who interact with each other, again, eliminates the possibility of *resolvers* coordinating an attack on *secondaries* as each *secondary* responds with the use of different keys, thus produces different and unrelated proofs, which cannot help *resolvers* extract any information from the *secondaries*, without explicitly querying for it.

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 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. [56], 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. [16], which we describe in more details in Section 4.4. Agrawal, Boneh and Boyen also offer two HIBE constructions [4,3] 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.

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

[4] Note that we allow to leak the information $f(R)$ in the execution of our protocol, so this information can be added as an output to the ideal process (i.e. if we allow leaking $|R|$ the ideal process should leak it too so that the two views described above will truly be indistinguishable).

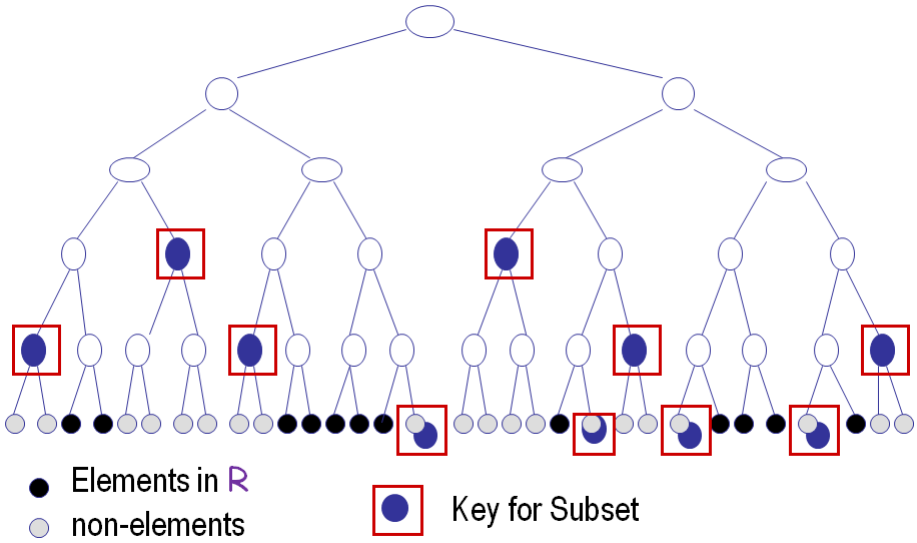

Fig. 2. A full binary tree that represents a set R and its set of keys K.

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 [34]. 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. [16] 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. [16].

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 3. In a recent paper, Lewko and Waters [51] 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 $O(r \log \frac{|U|}{r})$ [56].

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 message 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 $\frac{\varepsilon}{2}$ [5]. 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.

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. □

[5] 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.

On Transferability of Responses and Using Hierarchical Identity Based *Signatures*

We can also use Hierarchical Identity Based *Signatures* (HIBS for short) instead of encryptions. The difference between the two approaches is that instead of encrypting messages you can generate signatures and verify them. This way a *resolver* doesn't have to issue a random challenge, it can just query for x and get in response a signature on the element x (under the secret key corresponding to $h(x)$), saying it is not in R. This makes *secondary* responses to queries transferable, meaning a *resolver* could prove to another *resolver* that $x \notin R$ by sending it the signature it got when it queried that element.

Another way we can get transferability of responses is instead of using random challenges by the *resolvers*, we could just have them query x and in case $x \notin R$ have them get the secret HIBE key for $h(x)$, i.e. $SK_{h(x)}$. This way the *resolvers* could encrypt random challenges and decrypt them by themselves to verify the correctness of the response. This way they could transfer the response to other *resolvers* to prove to them that $x \notin R$. Notice that when we use this technique we shift some of the computational load from the *secondaries* to the *resolvers* as now *secondaries* don't need to decrypt any challenges, just generate a key, which they did before anyway, but now the *resolvers* need to decrypt a challenge which they issued, to make sure they got a valid key and a correct response.

4.4 HIBE Construction by Boneh, Boyen and Goh

We describe the construction by Boneh et al. [16] 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 (see Section 11.3 for details). 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[6]. 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 $ID^* = (I_1, \dots, I_m) \in (\mathbb{Z}_p^*)^m$ and a private key of its ancestor $ID = (I_1, \dots, I_k)$ $(m > k)$ do the following in order to generate a properly distributed key:
  If $SK_{ID} = (g_2^\alpha \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^{r'}, g^{r'}, h_{k+1}^{r'}, \dots, h_\ell^{r'}) = (a_0, a_1, b_{k+1}, \dots, b_\ell)$ then choose uniformly at random $t \in \mathbb{Z}_p$ and output:

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

[6] Note that we could use a tree with smaller depth (i.e. decrease ℓ) and map more than one bit of an element to every coordinate in the hierarchy of identities. This modification though, will result in more work for the *primary*, as it will increase the size of the set of HIBE keys K and the computation and size of Aux (defined in the HIBE setup algorithm). Asymptotically the efficiency of the scheme remains about the same, unlike the size of the keys which grows, thus we use binary identities.

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, defined in Section 11.4) 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 Systems Based on One-Time Signatures

In this section we describe a PSR system that works along the line of the HIBE construction in that they both have a full binary tree structure, where we remove all paths from the root to leaves corresponding to elements in R. This structure is closely related to that of punctured pseudorandom functions [70]. We allow *secondaries* to generate chains of signatures in this construction (which constitute a proof of non-membership) and secret decryption keys in the HIBE construction, only for elements outside of R, where

in a punctured PRF, one can compute the PRF on all values besides one. We will use one-time signatures, which are signatures that are infeasible to forge for any polynomial time adversary who witnesses at most one signature of its choice (signatures and k-time existential security are defined in Section 11.2). In fact we will need to sign at most two times with each key, so we can use one-time signatures and double the keys. A proof of non-membership for an element $x \notin R$ will be a chain of signatures from the root of the full binary tree to the leaf corresponding to $h(x)$, where each node on the path signs the public key of its descendant and the public key of the root is part of the scheme's public key. In order to remain consistent (give the same proofs at different times, or for different *secondaries*), we use the GGM pseudorandom function [38]. For every node x in the full binary tree, the PRF produces three values: labels for the two children of x and randomness used to generate the secret and public keys corresponding to x.

We describe the PSR system in detail:

Setup - Use the setup algorithm for the (regular and consistent) signature scheme and obtain (PK_{sig}, SK_{sig}, h) where h is a universally one way hash function (UOWHF) that maps elements from U to $\{0,1\}^{\ell}$. Also generate a GGM pseudorandom function [38] $F = (F_L, F_R, F_S)$ which maps a label of a node x to three strings: a label for each of x's children (F_R for the right child and F_L for the left) and randomness used to generate the cryptographic keys for the one-time signature scheme (F_S for the one-time signature)[7]. Use the setup algorithm for the one-time signature scheme and obtain the secret and public keys for the root of the full binary tree of depth ℓ, denoted as (PK_{root}, SK_{root}) (generate keys which are secure for two signatures). As for the randomness used for the generation of these keys, draw uniformly at random a label for the root L_{root}, invoke F_S on that label to generate the randomness for the keys. Set the public key to be $PK = (PK_{sig}, PK_{root}, h)$.

Now generate the forest of full binary trees, as we did for the HIBE, 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 a chain of signatures, along the path from the root of the full binary tree (of depth ℓ), ending with t_j. The first signature in the chain is signed by SK_{root} and each node signs with its secret key, the public key of its descendant coupled with its encoding (i.e. for the descendant x, sign (x, PK_x)). The generation of keys for a node x, child of y (where L_y is its label), is done using the setup algorithm for the one-time signatures, which uses randomness generated by $F_S(F_L(L_y))$ in case x is a left child and $F_S(F_R(L_y))$ otherwise. Denote the chain of signatures leading up to t_j as c_j, the secret key corresponding to c_j as SK_j and the label for t_j as L_j and set $K = \{t_j, L_j, c_j, SK_j\}$.

In order to produce proofs of membership, sign every element $x_i \in R$ with its value: $s_i = (Sign_{SK_{sig}}(x_i, v_i), (x_i, v_i))$. Note that each node in U either has a signature proving its membership in R, or it has a chain of signatures ending at one of its ancestors, which can be completed online. Set the secret key to be $SK = (K, \{s_i\}_{i=1}^{r}, F)$.

Verify - Gets an element $x \in U$ and the public key and initiates an interactive protocol with a *secondary* by sending it x in the clear. 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.

If it gets a chain of ℓ signatures, it verifies the first signature with PK_{root} and every next signature is verified with the public key which was decrypted in the previous signature in the chain. Each signature must contain the public key of the next node along the path from the root to the leaf and the node's encoding (to verify the correct path). Only if all signatures verify correctly and the path of signatures ends in $h(x)$, then it accepts that $x \notin R$ and returns 1. Otherwise it returns 0.

Prove - Gets the public and secret keys and also x from a *resolver*. If there exists a signature s_i for which $x_i = x$, then return s_i. Otherwise the secret key contains K, contains a chain of signatures from the root to a node which is an ancestor of $h(x)$ (or $h(x)$ itself), plus the secret key corresponding to that node and its label. Complete the chain of signatures by signing the next node in the path, coupled with its public key (generated using randomness from F and its label) until the end of the chain at node $h(x)$. Send the chain of signatures back to the *resolver*.

Now we claim that the resulting scheme is a prefect ZK PSR system.

[7] Note that we use the GGM implementation of a PRF as it is the most natural choice when dealing with labels of binary trees. The choice of a seed for the PRF determines the labels of the full binary tree of depth ℓ, thus we use the full binary tree structure both for the GGM PRF and for the tree of signatures.

Proof sketch. Perfect completeness holds as for every $x_i \in R$ there is a precomputed signature s_i in the secret key which will always be verified successfully. Regarding elements $x \notin R$, the *secondaries* can generate a valid chain of signatures for each such element as by definition of K, it contains a secret key for a node which is an ancestor of $h(x)$ and a chain of signatures leading up to it, from which it can complete the chain of signatures which will be verified successfully by a *resolver* with probability 1.

Soundness also holds as if one can prove for $x \notin R$ that $x \in R$ then it will violate the existential unforgeability of the (regular) signature scheme. If one could prove for $x \in R$ that $x \notin R$ then it means it managed to forge a one-time signature, since we defined K not to contain any secret key for an ancestor of $h(x)$, for $x \in R$. But as we defined the scheme, each secret key is used to generate at most two signatures (one for each of its children), thus the forgery contradicts the assumption that the signature scheme is secure against two signatures, thus proving that the system is sound. Note that because each node's secret key is generated using the randomness of F over its parent's *secret* label (*resolvers* are not aware of the nodes' labels, just their encodings), *secondaries* don't have any label for an ancestor of $h(x)$ for $x \in R$, so they cannot produce the secret keys for any node in the chain of signatures that ends at $h(x)$, thus *secondaries* cannot forge a proof for the false statement $x \notin R$.

Perfect ZK is proven similarly to the HIBE construction. The simulator again runs most of the setup algorithms which the PSR setup algorithm uses. It generates the GGM PRF F, the hash function h and chooses a random label L_{root} for the root of the tree. It then produces a pair of keys for the (regular) signature scheme, (PK_{sig}, SK_{sig}) and generates the keys for the one-time signature scheme (PK_{root}, SK_{root}) (using the randomness of $F_S(L_{root})$), corresponding to the root of the full binary tree. The simulator then sets the keys to be $PK^* = (PK_{sig}, PK_{root}, h)$ and $SK_{SIM} = (SK_{sig}, SK_{root}, F, L_{root})$. Like the case for the HIBE construction the precomputed signatures $\{s_i\}$ are replaced with the signature key SK_{sig} and the set of keys K is replaced with a secret key which can generate proofs of non-membership for any element in the universe, this time it is SK_{root}, while the HIBE construction used the master secret key for the HIBE. In order to generate proofs for $x_i \in R$, the simulator generates the signatures $s_i = (Sign_{SK_{sig}}(x_i, v_i), (x_i, v_i))$ online. For elements $x \notin R$ the simulator simply generates the chain of signatures starting from the root of the tree ending at the leaf corresponding to $h(x)$, exactly like the Prove algorithm would do, using randomness generated by the PRF F starting with the root label, L_{root}.

The resulting view of the adversary communicating with SIM will be identically distributed to the view of the adversary communicating with a real PSR system. The fake public key PK^* is generated the same way the original keys are generated. Proofs regarding $x \in R$ are generated by the same algorithm, just online instead of before hand, but the distribution of the signatures remains the same. Proofs regarding $x \notin R$ are identically distributed for both views, using the secret root label L_{root} and F, the simulator can produce a proof of non-membership for every element in the universe U, thus resulting in the same distribution of proofs for $x \notin R$ as the distribution of a real PSR system, which also chooses a PRF and such a label, which determine the entire set of secret keys and labels for the tree. The only difference between those proofs is that a real PSR system generates a part from each chain of signatures before hand and the *secondary* completes the chain online, while the simulator generates the whole proof online, but the resulting chains are identically distributed. Either way once the function F and the label L_{root} are set the entire set of keys for the tree is set and both the function and the label are determined at the setup phase, randomly drawn from the same distribution. Thus we get two identically distributed views, which results in perfect ZK. □

Now as we can construct both types of signatures (one-time and regular) from universally one way hash functions (UOWHF) [61], 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 [68]. PSR systems imply identification schemes, as shown in our companion paper [36], which in turn imply the existence of one-way functions, as shown by Impagliazzo and Luby [45] (see also [44]).

Note that these are all black box constructions, which give us 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 (see Figure 3 for clarification).*

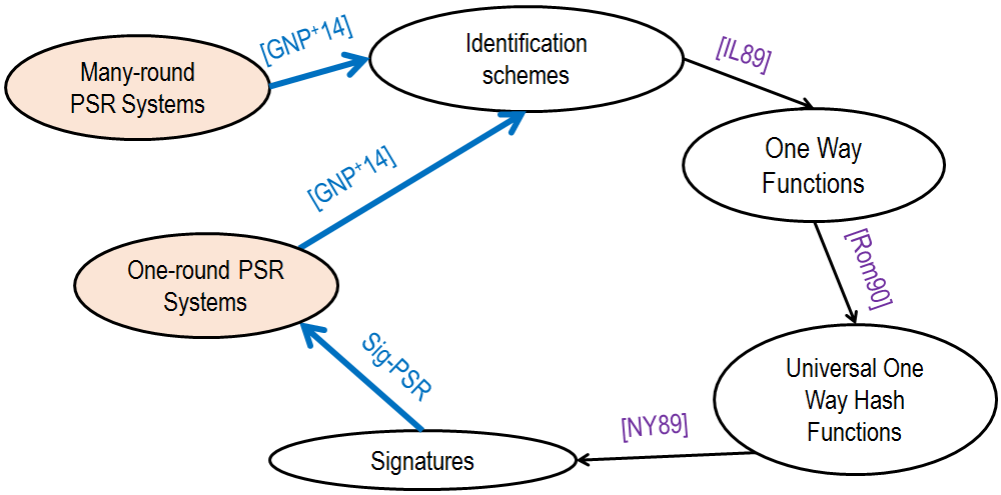

Fig. 3. Each node represents a cryptographic primitive and an arrow means that the existence of one primitive implies the existence of the other, as proved by the citation above it ([36,45,68,61]). Sig-PSR represents the one-time signatures PSR construction, presented in this section.

Chase et al. [25] proved that interactive ZKS and collision resistant hash functions (CRH) are existentially equivalent, i.e. you can construct one from the other. Simon [75] 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).*

Note that if we have an efficient one-time signature scheme (in terms of the complexity of the signature and verification algorithms), then this PSR system can be quite efficient as the time it takes a *secondary* to produce a response is dominated by at most $O(\log |U|)$ signatures and the time it takes to verify is dominated by $O(\log |U|)$ verifications. Thus an efficient signature scheme with this very weak security requirement can produce an efficient and practical PSR system.

6 Cuckoo Hashing Based Construction of PSR Systems

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. We describe a construction based on cuckoo hashing, a scheme first introduced by Pagh and Rodler [64]. We will think of cuckoo hashing in the static case, where the set R is fixed in advance, as is the case with PSR systems. A cuckoo hashing scheme uses two hash functions (F_1, F_2), which map elements in the universe, into two tables T_1 and T_2 of size $(1+\varepsilon)r$. The *primary* starts by mapping the r elements of R into the tables T_1 and T_2, where each element $x \in R$ is placed either in T_1 in location $F_1(x)$ or in T_2 in location $F_2(x)$. As collisions can always cause this task to be impossible for some chosen set R and functions (F_1, F_2), the *primary* may need to draw the functions (F_1, F_2) a few times, until it finds ones that fit the set R (as we shall see, in our case the probability of redrawing will be negligible). When the *primary* finds proper functions, it commits to the placements of the set R in the tables. In order to prove membership in the set for a cuckoo hashing scheme the prover proves to the verifier

that x is placed in one of its two possible placements (either in $T_1(F_1(x))$ or $T_2(F_2(x))$), while in order prove non-membership it proves that x is not placed in either of its two possible positions. We will use this technique for our non-membership proofs.

There are different variations on cuckoo hashing which try to improve its success probability and efficiency. For example, one may use tables with buckets which are larger than 1 in order to reduce the probability of choosing bad functions for the set of elements. We will be interested, for our denial of existence mechanism, in the variation which uses a "stash", where the elements that could not be placed in the tables, due to overcrowding, are kept. This reduces the probability of having to choose new functions for the cuckoo hashing, even when using a relatively small stash, as shown by Kirsch, Mitzenmacher and Wieder [49]. As a non-membership proof for $x \notin R$, *secondaries* will prove that x is not in either of the two locations of the table *and* that x is not in the stash (by proving that none of the elements in the stash are x). Cuckoo hashing has been used in cryptography in the past, for example [12], which uses cuckoo hashing for a hardness preserving reduction from non-adaptive to adaptive PRFs and in works on constructing Oblivious RAMs [66,41].

6.1 Cuckoo Hashing with a Stash

We give a brief description of how cuckoo hashing works and its different properties. Roughly speaking, in a cuckoo hashing scheme we draw two universal hash functions $F_1, F_2 : U \to [\ell]$, where $\ell \geq (1+\varepsilon)r$ for some constant ε, and r is the number of elements to be inserted into the cuckoo hash. A lookup for x in the table is simply checking if $T_1(F_1(x)) = x$ or $T_2(F_2(x)) = x$. In order to put the set R in the cuckoo hashing, one simply inserts the elements one by one. An insertion of x is as follows: set $y = T_1(F_1(x))$ and $T_1(F_1(x)) = x$, if $y = \perp$ (empty) finish, else set $z = T_2(F_2(y))$ and $T_2(F_2(y)) = y$, keep this procedure going until all elements are in the tables (if it is possible). It is clear that not all functions (F_1, F_2) will be able to properly place the set R in both tables. Consider the bipartite cuckoo hash graph $G_{F_1,F_2,R}$ that results from the two functions (F_1, F_2) and the set R, by setting the nodes to be the cells of the tables T_1 and T_2, i.e. 2ℓ nodes and the edges to be $E = \{(F_1(x), F_2(x)) | x \in R\}$, then if and only if $G_{F_1,F_2,R}$ has at most one cycle in every one of its connected components, we can place the set successfully in the cuckoo hash. See Figure 4 for an example.

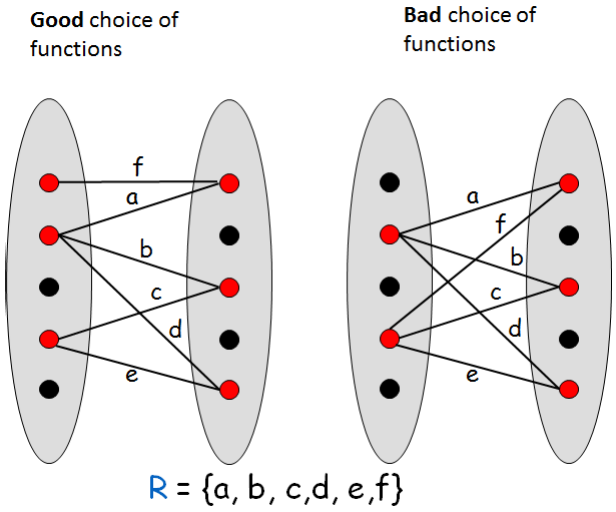

Fig. 4. Two choices of functions, where the left one suits the set R and the right one doesn't, since it contains more than one cycle in its connected component.

Pagh and Rodler [64] showed that the probability of choosing bad functions when $\ell \geq (1+\varepsilon)r$ and F_1 and F_2 are drawn from an $(O(1), O(\log r))$-universal hashing family[8] is $O(\frac{1}{r})$, while the entire insertion process takes $O(r)$ expected time. Note that if we use cuckoo hashing as described above as a denial of existence mechanism, then a malicious *resolver* could learn about the set R some information just by the choice of the functions F_1 and F_2 (that are public), as about an $O(\frac{1}{r})$ fraction of the functions will not produce a valid assignment for R.

In order to diminish the amount of information we release to *resolvers* about the set R, we use cuckoo hashing with a stash. It is a cuckoo hashing scheme where we keep a stash of elements that we could not fit in the tables T_1 and T_2. For example, if the bipartite graph for the set R and functions F_1 and F_2, after we remove 3 edges from it, has at most one cycle in every one of its connected components then a stash of size 3 will suffice for us to commit to the set R successfully with the functions (F_1, F_2). This makes our lookup procedure longer, as now one also needs to check the stash and not only the tables T_1 and T_2, to verify whether an element is in the cuckoo hash. Kirsch, Mitzenmacher and Wieder [49] show that the probability that the stash is larger than s is bounded by $O(r^{-s})$, albeit in their analysis they require the hash functions to be fully random, instead of being drawn from an $(O(1), O(\log r))$-universal hashing family. Luckily, Aumüller et al. [6] addressed that issue and suggested a construction of hash functions which suffices in order to get an $O(r^{-s})$ probability of failure. An alternative is to use pseudorandom functions for F_1 and F_2. Usually releasing the secret key of the PRF is dangerous, but here we are interested in a property that can be tested on a given set R. If the function fails on it (whereas a truly random function would not), then we have a distinguisher for the PRF.

Thus, if we take a stash of size $\log_r |U|$ we will get a negligible probability that functions F_1 and F_2 do not properly place the set R in the tables plus stash. This means that if we use a cuckoo hash with a stash as our denial of existence mechanism, we will release significantly less information about the set R than with a regular cuckoo hash. Note that by doing so we reduced the problem of proving non-membership for the set R to the set S and as the bound on the stash is fairly small compared to the size of the original set R, it reduces the problem to be proportional in its efficiency to a much smaller set, i.e. S instead of R.

6.2 Construction of PSR Systems from Cuckoo Hashing with a Stash

We describe our cuckoo hashing with a stash based PSR construction and prove it to be a PSR system with f-ZK (for $f(R) = |R|$). Except for the cuckoo hashing with a stash, we will also need to use commitments with inequality proofs and a method for proving non-membership for a fixed set. We will require both protocols to be complete, sound and indistinguishable in the case we commit to "dummies" instead of the real set. We describe both required primitives next.

A commitment scheme is comprised of two phases: *commit* and *reveal*. In the commit phase, you generate a commitment c to a string x with secret information s. In the reveal phase we will only require that using (c, x, s), one can prove interactively for an element $y \neq x$ that the committed value is not equal to y[9]. We require the commitments to be *hiding*, meaning that for every $x \neq x'$ and uniformly at random chosen secrets s, s', the ensembles $\{Commit(x, s)\}$ and $\{Commit(x', s')\}$ are identically distributed (perfect hiding), statistically close (statistically hiding) or computationally indistinguishable (computationally hiding). We also require the commitments to be computationally binding, meaning that it is infeasible for a polynomial time attacker which gets a commitment c to x with the secret information s ($c = Commit(x, s)$), to find (x', s') such that $Commit(x', s') = c$. In our case the binding requirement is a bit weaker: given that the *primary* is choosing the commitment and giving it to the *secondary*, all we need is that for a properly chosen commitment it is hard to find a different opening. A commitment may also be to a 'dummy' element, in which case all x's are different from it.

[8] A (c, k)-universal hashing family $\{h_i : U \to L\}_{i \in I}$ is a family of hash functions such that for every k distinct elements $x_1, \ldots, x_k$, any $y_1, \ldots, y_k$ and a uniformly random $i \in I$ it holds that $Pr[h_i(x_1) = y_1, \ldots, h_i(x_k) = y_k] \leq \frac{c}{|L|^k}$.

[9] Usually, one proves the value committed, but as we want to prove non-membership we do not require this feature in our commitment scheme.

Our commitment inequality proofs, which get a commitment C_x to an element x with secret information s and an element y and prove that C_x is not a commitment to y, have to satisfy three properties: completeness, soundness and indistinguishably. Completeness means that when the prover and verifier both act honestly then the protocol succeeds with overwhelming probability while the soundness condition states that a prover cannot convince a verifier to believe in a false statement with overwhelming probability. The indistinguishability requirement is that an adversary who chooses x and x' such that $x' \neq x$, cannot distinguish between a proof of inequality of x to a commitment to x' and a proof of inequality of x to a commitment to the "dummy" element. We note that this is a weaker requirement than ZK which implies indistinguishability.

The second protocol we require is a *fixed set non-membership proof system*. The only difference between this protocol and the commitment scheme is that we commit to a set instead of a single element. The *primary* generates two keys (secret and public) for the set S, where a *secondary* could use the secret key to prove to a *resolver*, with knowledge of the public key and no knowledge about the set S (besides maybe its cardinality), that an element x is not in the set, without revealing any elements in the set S. Our requirements from the protocol are identical to those of the commitment scheme: hiding (cannot tell the difference between different sets), binding and soundness (both requirements amount to it being infeasible to cheat on the set), completeness (honest execution results in the correct conclusions) and indistinguishability between non-membership of any set (even a set known to the distinguisher) and one consisting of only dummies.

Armed with a cuckoo hashing scheme, a commitment scheme with interactive proofs of inequality and a fixed set non-membership proof system, we describe the PSR system.

Setup. Compute the cuckoo hash with a stash as described above. Generate the keys (PK_S, SK_S) for the fixed set non-membership proofs with the set S, where S is the stash. Generate parameters for the commitment scheme, PK_{com}. Run the setup algorithm for the signature scheme to obtain (PK_{sig}, SK_{sig}) (we use consistent signatures which produce the same signature on the same message). Generate the signatures $s_i = (Sign(x_i, v_i), (x_i, v_i))$. If $x \in R$ is in table T_i in cell $j = F_i(x)$ in the cuckoo hash, then set $x_{i,j} = x$ and generate a commitment $C_{i,j}$ with secret information $r_{i,j}$: $C_{i,j} = Commit(x_{i,j}, r_{i,j})$. If cell j in table T_i is empty generate a dummy commitment $C_{i,j}$, i.e. draw uniformly at random an element $x_{i,j}$ and secret information $r_{i,j}$, check that $F_i(x) \neq j$[10] and compute $Commit(x_{i,j}, r_{i,j}) = C_{i,j}$. For every such commitment $C_{i,j}$ (dummy or regular), compute $s_{i,j} = (Sign(C_{i,j}, i, j), (C_{i,j}, i, j))$. Note that $x_{i,j}$ and $r_{i,j}$ are the value and secret of the commitment at cell j of table i.

Set the public key of the scheme to be $PK = (F_1, F_2, PK_{sig}, PK_S, PK_{com})$ and the secret key to be $SK = (\{s_{i,j}\}_{j=1,i\in\{1,2\}}^{\ell}, \{s_i\}_{i=1}^{r}, \{C_{i,j}, x_{i,j}, r_{i,j}\}_{j=1,i\in\{1,2\}}^{\ell}, SK_S)$.

Prove. Gets x, if $x = x_i \in R$ then return the signature s_i. If $x \notin R$ then locate commitments $C_{1,F_1(x)}$ and $C_{2,F_2(x)}$, send signatures $s_{1,F_1(x)}$ and $s_{2,F_2(x)}$ to the *resolver* and prove (using the inequality proofs) that both commitments are for a different value than x using $(x_{1,F_1(x)}, r_{1,F_1(x)})$ and $(x_{2,F_2(x)}, r_{2,F_2(x)})$ respectively. Also prove that $x \notin S$ using the fixed set non-membership proof system and its secret key SK_S.

Verify. Sends x to the *secondary* and gets one of two possible responses. For a membership proof it gets (s, x, v) where s is supposed to be a signature on the pair (x, v) and if it verifies correctly that s is a valid signature on (x, v), it accepts and outputs 1. For a non-membership proof of $x \notin R$, the *secondary* gets signatures $s_{1,F_1(x)}$ and $s_{2,F_2(x)}$ and engages in three interactive proofs, two for the two commitments in the cuckoo hash in positions $(1, F_1(x))$ and $(2, F_2(x))$ in the tables and one that proves $x \notin S$ (where S is not known to the *resolver*). The *resolver* verifies that all interactive proofs are valid using its public keys for the protocols (PK_{com} for commitments and PK_S for fixed set non-membership) and also checks that the commitments for which the *secondary* proved inequality are the signed commitments in the correct positions (using the signatures which it gets from the *secondary*). If all checks pass successfully the *resolver* accepts and returns 1, else it returns 0.

We now argue that this construction is indeed a PSR system with f-ZK, for $f(R) = |R|$.

Proof sketch. In order to prove this construction to be a PSR we need to prove it to be complete, sound and f-ZK (for $f(R) = |R|$).

[10] We check that so we could prove non-membership for x, if $F_i(x) = j$ we could not prove that x is not in table T_i, due to the soundness property of the commitment scheme.

Completeness. Elements $x_i \in R$ can always be proven successfully as the precomputed signature s_i is part of the secret key, thus known to the *secondary*, while the public key for the signature scheme is part of the PSR's public key, which is why it will be verified successfully. Regarding elements $x \notin R$, proving the commitments' inequalities succeeds with overwhelming probability, as we know we committed to a different value than x in the places $T_1(F_1(x))$ and $T_2(F_2(x))$ and the completeness guarantee of the commitment scheme guarantees our overwhelming success. Proving the non-membership of x in S is also guaranteed to succeed with overwhelming probability due to the completeness condition of the fixed set non-membership proofs we use.

Soundness. Proving false statements regarding elements $x \notin R$ requires the *secondary* to generate a valid signature on x and some value v without the knowledge of the secret key for the signature scheme. In order to prove a false value for an element $x \in R$, an adversary also has to generate a valid signature without knowing the secret key. According to the existential unforgeability of the underlying signature scheme we know this is not possible with more than a negligible probability of success. Proving false statements regarding $x \in R$ requires a cheating *secondary* to do one of three things. If x is placed in the stash then it has to prove $x \notin S$, which is a false statement on the set S violating the soundness property of the fixed set non-membership proofs for S. If x is placed in table T_i of the cuckoo hash, then the *secondary* can try to forge a signature for a different commitment, to replace the commitment to x in position $F_i(x)$ of table T_i, but due to the existential unforgeability of the signature scheme this is infeasible for the *secondary*. The *secondary* can also try to prove that the commitment $C_{i,F_i(x)}$ is not a commitment to x, which is false and thus cannot be done with more than a negligible probability of success, unless one knows x', r' for which $Commit(x', r') = C_{i,F_i(x)}$, according to the soundness property of the commitment scheme. But according to the binding property of the commitment scheme, it is infeasible to find such (x', r') thus making it infeasible to prove such a false statement. Either way we see that a *secondary* doesn't have more than a negligible probability to cheat a *resolver*, thus making the system sound.

f-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 generates the keys for the signature scheme (PK_{sig}, SK_{sig}) and also generates public parameters for a commitment scheme PK_{com}. SIM gets $f(R) = |R|$ and the universe U, so it knows the size needed for the tables in the cuckoo hash (each table of size ℓ) with a stash (of size $\log_{|R|} |U|$) and thus knows from which distribution it has to draw the functions (F_1, F_2). Then it continues to generate fake keys for the fixed set non-membership proofs, by committing to S, a set of dummies, i.e. it chooses the elements uniformly at random from the universe U and obtains (PK_S, SK_S). It then fills the tables of the cuckoo hash with random commitments, using PK_{com} to generate commitments to random elements $x_{i,j}$ with secrets $r_{i,j}$: $C_{i,j} = Commit(x_{i,j}, r_{i,j})$ for $i = 1, 2$ and $j = 1 \dots \ell$. It checks before committing to each element that $F_i(x_{i,j}) \neq j$ so we could prove inequality for all elements which map to this commitment. It signs each commitment just as the *primary* does: $s_{i,j} = (Sign(C_{i,j}, i, j), (C_{i,j}, i, j))$. It sets the fake public key to be $PK^* = (F_1, F_2, PK_{sig}, PK_S, PK_{com})$ and the fake secret key to be $SK_{SIM} = (\{s_{i,j}\}_{j=1,i\in\{1,2\}}^{\ell}, SK_{sig}, \{C_{i,j}, x_{i,j}, r_{i,j}\}_{j=1,i\in\{1,2\}}^{\ell}, SK_S)$.

When queried on an element x the simulator asks its oracle on x. If $x \in R$ and its value is v it generates $s_x = (Sign(x, v), (x, v))$ using the secret signature key SK_{sig}. When $x \notin R$ the simulator uses PK_{com} to prove inequality for the commitments positioned at $T_1(F_1(x))$ and $T_2(F_2(x))$, after sending the relevant signatures $(s_{1,F_1(x)}, s_{2,F_2(x)})$, like a *secondary* does. The committed values in both positions cannot be x because before committing to each random element $x_{i,j}$ we checked that $T_i(F_i(x_{i,j})) \neq j$. The simulator then proves that the element x is not in S, assuming it didn't choose x as one of the dummy elements in the set. If the simulator had committed to x, it stops as it failed to produce an indistinguishable view. Note that as we chose the dummies for the set S uniformly at random from U and S is of size $\log_{|R|} |U|$, the probability of querying such an element is bounded by $\frac{q(k)\cdot\log_{|R|} |U|}{|U|-\log_{|R|} |U|}$, where $q(k)$ is the polynomial bound on the amount of queries an adversary can issue, and this probability is negligible (as $|U|$ is exponential in k, while $|R|$ is at most polynomial).

We now claim that the two views generated by the simulator and a real system are indistinguishable. The public key PK^* is generated by the same algorithms the real system uses, but for dummy elements instead

of the real set R, but due to the indistinguishability property of both schemes (commitments and the fixed set), a distinguisher can't tell the difference. The dummy commitments we generate are indistinguishable from real commitments to the elements of R due to the hiding property of the commitment scheme. Proofs regarding $x_i \in R$ are signatures coupled with their values $(Sign(x_i, v_i), (x_i, v_i))$, generated the same way the original proofs are computed 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, since the signature scheme we use produces the same signatures on the same messages.

Proofs for $x \notin R$ are comprised of the inequality proofs for the commitments and the non-membership proofs for the stash. As we required indistinguishability form both proof systems, we know that even when we committed to dummies, a distinguisher that knows the set R could not distinguish between the two inequality/non-membership proofs for the set of dummies and the real set R. For the fixed set non-membership proofs, we have a probability of failure, but as we mentioned it is negligible. As we mentioned before, the fact that we use a stash of size $\log_{|R|} |U|$ means that only for a negligible fraction the functions (F_1, F_2) the set R will not fit in the cuckoo hash with a stash. Thus the choice of "bad" functions can fail us in generating an indistinguishable view from a real system with at most negligible probability. Note that this makes our ZK property at most statistical, even if both the commitment scheme and the fixed set non-membership proofs have perfect indistinguishability, as a distinguisher who knows R can always check if the functions F_1, F_2 fit the set R. Thus a distinguisher cannot distinguish the simulation from a real execution. □

6.3 Implementations for Commitments and Fixed Set Non-Membership

We suggest Pedersen commitments [65] as the commitment scheme for the PSR and a scheme that uses a generalization of the Feige-Fiat-Shamir identification protocol [31] as the fixed set non-membership proofs for a predetermined set. We require the same properties from both schemes, so it is not unreasonable to expect a scheme to implement both primitives, but due to the fact that we need to commit to only one set, as opposed to many commitments to elements, we use different implementations. The fact that we reduced the problem of set non-membership from the set R to the much smaller stash S, for which we commit only once, gives the *primary* the ability to use a shared random string (which will be proportionate in size to S), or instead incorporate it in the public key. Doing the same for the set R would have been inefficient.

Pedersen Commitments

We describe Pedersen's commitments [65] with ZK proofs of inequality briefly and specify their properties.
Setup. Generates large primes p, q such that q divides $p - 1$. Set g to be a generator for $\mathbb{G}$, a subgroup of order q of $\mathbb{Z}_p^*$. Choose a random integer $a \in \mathbb{Z}_q$ and set $h = g^a$. p, q, g, h are public while a is secret to all parties (i.e. even the prover can't know a).
Commit. In order to commit to a value $x \in \mathbb{Z}_q$, one simply draws uniformly at random $r \in \mathbb{Z}_q$ and computes $C_x = g^x h^r$.
Interactive ZK proof of equality. We use the ZK proof of knowledge for Pedersen commitments based on the adaptation of Schnorr's identification protocol [72], which is proven to be complete, sound and ZK. At every step of the interactive proof we specify in parenthesis the added actions we need to take in order to prove simultaneously that $C_x = g^x h^r$ and $B_x = k^x \ell^r$ in ZK, which we will use for the inequality proofs.

1. The prover which knows $C_x = g^x h^r$ chooses uniformly at random $y, s \in \mathbb{Z}_q$ and computes $D = g^y h^s$ and sends D to the verifier. (Compute $E = k^y \ell^s$ and send it to the verifier as well).
2. The verifier chooses uniformly at random $e \in \mathbb{Z}_q$ and sends it as a challenge to the prover.
3. The prover computes $u = y + ex$ and $v = s + er$ and sends them back to the verifier.
4. The verifier accepts the proof if and only if $g^u h^v = DC_x^e$. (The verifier also checks that $k^u \ell^v = EB_x^e$).

Interactive ZK proof of inequality. For a given commitment $C_x = g^x h^r$ (where x, r are known to the prover) and a given y, we can use the ZK proof of inequality for discrete logarithms suggested by Camenisch and Shoup [21] to prove that $x \neq y$.

1. The prover chooses uniformly at random $s, t \in \mathbb{Z}_q$ and computes $C = (\frac{h^x}{h^y})^{st}$. It then computes $A = h^{rt}$ and $B = g^{xt}$ and sends (A, C, t) to the verifier.
2. The verifier computes $\frac{C_x^t}{A} = \frac{(g^x h^r)^t}{h^{rt}} = g^{xt} = B$.
3. The prover proves that it knows integers (a, b) such that: $C = h^a(\frac{1}{h^{ty}})^b$ and $1 = g^a(\frac{1}{B})^b$ ($a = xst$ and $b = s$), using the simultaneous proof of equality described above.
4. The verifier accepts the proof if and only if it accepted the interactive simultaneous proof of equality and $C \neq 1$.

Correctness follows as from $1 = g^a(\frac{1}{B})^b$ we can conclude that $a = b \log_g B$. By substituting a in the second equation we get: $C = h^a(\frac{1}{h^{ty}})^b = (\frac{h^{\log_g B}}{h^{ty}})^b$ and as we checked that $C \neq 1$ it means that $\log_g B \neq ty$. But $\log_g B = tx$ which proves $x \neq y$. This protocol requires a total of 9 modular exponentiations and 3 modular multiplications from the prover and 8 modular exponentiations and 5 modular multiplications from the verifier.

Pedersen's commitments are proven to be perfectly hiding, meaning that for a given commitment c, every value x is equally likely to be the value committed in c. They are computationally binding under the discrete logarithm assumption (see Section 11.4), meaning that a prover that can find (x, x', r, r') such that $Commit(x, r) = Commit(x', r')$, can compute the discrete logarithm of h: $a = \log_g h$, which is a secret. The proof of inequality is both complete, sound and ZK (which implies indistinguishability) as shown by Camenisch and Shoup [21].

Fixed Set Non-membership using the Feige-Fiat-Shamir Protocol

The idea for proving non-membership with a fixed set is to have a set of secrets corresponding to some public and fixed random collection. Each element in the universe is assigned a certain subset of the secrets and the rule is that if $x \in S$ then the *secondary* knows *none* of the corresponding secrets. I.e. the *secondary* gets all the secrets that do not belong to any subset of elements in S. A good assignment of subsets to elements is such that for $x \notin S$ there is a least one secret not covered by elements of S. Thus to show that $x \notin S$ it is enough if the *secondary* shows that it knows *at least one* of the corresponding secrets.

In order to construct a protocol for a fixed set non-membership proof system we will use the set lower bound technique of Goldwasser and Sipser [40], to allow the prover to show that it knows a *large* fraction of the secrets as opposed to knowing none of them. The basic technique is to map the universe of secrets to a much smaller domain, and with knowledge of a large enough fraction of the secrets, a prover will still be able to prove it knows a secret to at least one preimage of every element in the function's range with overwhelming probability. Specifically a *resolver* will be asking to get a proof corresponding to some vector v, where it gets to pick the constraints on the vector (for a fixed matrix C it will be a vector t such that $Cv = t$). A *secondary* will succeed in its proof if it can find a vector it knows the proof for which stratifies the *resolver*'s constraints. A cheating *secondary* will know the proof for only one vector (which will satisfy the constraints with negligible probability) whereas a good *secondary* will know proofs to many vectors.

The subset assignment: We will generate $n = \log |U|$ universal pairwise independent hash functions $(\{h_i\}_{i=1}^n)$, which map the universe of elements to $2s$ distinct values (where s is the maximal size we allow the fixed set, $s = \log_r |U|$). The i^{th} hash function will map the elements in the universe to the integers: $\{2s \cdot i, \ldots, 2s \cdot (i+1) - 1\}$. Every integer $k = 2s \cdot i + j$ (where $i \in [n]$ and $j \in [2s]$) will be mapped by M to some hard to compute unique challenge $M(k) = c_{i,j}$, where for each such challenge there is a corresponding answer or secret, that the generator of the scheme can compute, denoted as $x_{i,j}$. The *primary* which generates all those challenges and can also produce their corresponding secrets will give *secondaries* the secrets $x_{i,j}$ corresponding to the challenges $c_{i,j}$ iff $M(h_i(S)) \neq 2s \cdot i + j$, meaning no element $x \in S$ is mapped by the i^{th} mapping to the j^{th} challenge, i.e. we give the secrets to every challenge that doesn't correspond to an element of S.

For every $x \notin S$ we get that the probability that such a mapping maps x and some element $y \in S$ to the same challenge is at most $\frac{1}{2}$, due to the pairwise independence of the hash functions and the uniqueness of

the mapping M. Thus in expectation for a specific $x \notin S$ a *secondary* will know the secrets to about $\frac{1}{2}$ of the n challenges which correspond to x (as it only knows secrets to challenges not in S). Using the Hoeffding inequality [42] (see Appendix 11.5) we reach the conclusion that with overwhelming probability (in n), for every $x \notin S$ the *secondaries* will know at least $\frac{n}{3}$ secrets for x out of n challenges. On the other hand for every element $x \in S$ the *secondaries* will not know even one secret out of the n challenges. Thus we would like to devise a system where knowing at least $\frac{n}{3}$ of the secrets will give *secondaries* the power to convince *resolvers* that the element in question is not in S while if they know none of the secrets, it is infeasible for them to convince a *resolver* that the element is not in S.

Our starting point is the Feige-Fiat-Shamir [31] identification protocol as the protocol for proving knowledge of a secret corresponding to some specific challenge, which is proven to be ZK. We present the original Feige-Fiat-Shamir identification protocol first, before extending it for our needs:

A trusted authority generates n secrets by assigning quadratic residues modulus an RSA-like modulus $N = PQ$ as public keys: $y_1, \ldots, y_n$. The corresponding secret keys are $x_1, \ldots, x_n$ where $y_i = x_i^2 \bmod N$. The prover then wants to prove to the challenger that it knows all the secret keys:

1. The prover draws uniformly at random $1 \leq a \leq N-1$ and sends $b = a^2 \bmod N$ to the challenger.
2. The challenger sends in response a challenge vector $v \in \{0,1\}^n$.
3. The prover computes $b_v = a \cdot \prod_{v_i=1} x_i \bmod N$ and sends it back to the challenger.
4. The challenger accepts the response as valid iff $b_v^2 = b \cdot \prod_{v_i=1} y_i \bmod N$.

We cannot use this protocol as is, because as we mentioned before, our *secondaries* will not know all n secrets for the elements outside of S, but will know at least $\frac{n}{3}$ of them with overwhelming probability. Notice that if the *secondary*, which picks a at step 1, can predict the vector v, then it can choose $b = \frac{a^2}{(\prod_{v_i=1} y_i)}$ and send at step 3 as a response $b_v = a$, which in turn will be verified successfully.

Thus we will generalize this protocol by having a set of global linear constraints on vectors $v \in \{0,1\}^n$ and having the *resolvers* choose the constraints on the vector. The *secondary* will in turn have to find a vector v which satisfies all the constraints and compute b_v such that $b_v^2 = b \cdot \prod_{v_i=1} y_i$. The probability of satisfying k constraints without knowing any of the secrets is 2^{-k} by trying to guess a vector to satisfy the constraints, thus we will use $\frac{n}{4}$ constraints in order for *secondaries* to be able to cheat with only a negligible probability.

The Mapping. If we want to use the Feige-Fiat-Shamir protocol as our underlying scheme, we will need a mapping M from the integers $[2sn]$ to quadratic residues modulus N. We are willing to use a fixed mapping, based on a shared random string, in order to reduce communication complexity. We interpret the shared string as a sequence of integers of length roughly $\log N$. But this may not be a straightforward mapping to integers modulus N, since not every integer modulus N has a square root. Denote by QR_N the set of quadratic residues modulo N.

Consider a large publicly known random string that defines a large table of random integers, which every party of the protocol knows. We use this table to map every $k \in [2sn]$ to two integers from the table denoted (z_1^k, z_2^k). We will generate our challenge for k from (z_1^k, z_2^k). We choose our modulus $N = PQ$ in the scheme to be a Blum integer, which means that we pick $P \equiv Q \equiv 3 \bmod 4$. For a prime $P \equiv 3 \bmod 4$ and every integer z it holds that either $z \in QR_P$ or $-z \in QR_P$. We have $z \in QR_N$ (where N is a Blum integer) iff $z \in QR_P$ and $z \in QR_Q$. It also holds for Blum integers that if $z_1, z_2 \in QR_P$ but $z_1, z_2 \notin QR_Q$ then $z_1 \cdot z_2 \in QR_N$. Thus if the challenge $k \in [2sn]$ is mapped to the integers z_1, z_2 we know that at least one of the following is a quadratic residue modulo N: $(z_1, z_2, -z_1, -z_2, z_1 \cdot z_2, -z_1 \cdot z_2)$, because if $z_1, z_2, -z_1, -z_2 \notin QR_N$, then at least two of them are in QR_P and not in QR_Q, which means their product is in QR_N. As the *primary* knows the factorization of N it will add 3 bits for every challenge $k \in [2sn]$ to the public key, specifying which of the six possible integers is a quadratic residue modulo N and will serve as the challenge.

Note that as all the integers in the table are random, the fact that an adversary gets to learn, say, that a challenge $c_{i,j} = z_1 \cdot z_2 \bmod N$ cannot help him, as if he draws a random integer $z_1 \in [N]$ and computes $z_2 = \frac{c_{i,j}}{z_1} \bmod N$ he can get the same outcome by himself. I.e. the distribution would be identical in this case.

The constraints and the generalization of the Feige-Fiat-Shamir protocol. Consider a constraints matrix $C = \{c_{i,j}\}$ with $\frac{n}{4}$ rows and n columns. Since we showed that the *secondary* knows at least $\frac{n}{3}$ secrets with overwhelming probability, we will assume w.l.o.g that the *secondary* knows secrets $x_1, \ldots, x_k$ (where $k \geq \frac{n}{3}$) and doesn't know the rest of the secrets. N and the challenges $y_1, \ldots, y_n$ are known to both parties. We claim that in the following protocol, the *secondary* will succeed with overwhelming probability:

1. The *secondary* draws uniformly at random $1 \leq a \leq N-1$ and a vector $v^2 = (v_{k+1}, \ldots, v_n) \in \{0,1\}^{n-k}$. It computes $b = \frac{a^2}{(\prod_{i>k:v_i=1} y_i)}$ and sends it to the *resolver*.
2. The *resolver* sends in response a target vector $t = (t_1, \ldots, t_{\frac{n}{4}}) \in \{0,1\}^{\frac{n}{4}}$.
3. The *secondary* tries to find a vector $v^1 = (v_1, \ldots, v_k) \in \{0,1\}^k$ such that the vector $v = (v^1, v^2)$ satisfies all the constraints ($Cv = t$). If it finds such a vector it computes $b_v = a \cdot \prod_{i \leq k:v_i=1} x_i$ and sends (v, b_v) to the *resolver*, else it fails.
4. The *resolver* accepts the response as valid if $Cv = t$ and $b_v^2 = b \cdot \prod_{v_i=1} y_i$.

Now in order to show a *secondary* succeeds in this protocol with overwhelming probability we need to show it can find a suitable vector in step 3 with overwhelming probability. We claim that a sufficient requirement for a *secondary* to find a vector to satisfy these constraints is that the matrix C^* is of full rank ($\frac{n}{4}$), where C^* is C restricted to the columns corresponding to the secrets the *secondary* knows, i.e. the first k columns.

We define the vector $t^* = (t_1^*, \ldots, t_{\frac{n}{4}}^*)$, where $t_i^* = t_i \oplus_{\{j>k:C_{i,j}=1\}} v_j$. If the matrix C^* is of full rank then we can find a vector v^1 for which $v = (v^1, v^2)$ satisfies all the constraints: $Cv = t$. Any solution to $C^*v^1 = t^*$ will be good for us and we can compute it efficiently. We will pick v^1 uniformly at random from the set of possible solutions. To see that this solution is indeed valid, for any row i we have

$$\sum_{j=1}^{n} C_{i,j} \cdot v_j = t_i^* + \sum_{j=k+1}^{n} C_{i,j} \cdot v_j = t_i^* + \oplus_{\{j>k:C_{i,j}=1\}} v_j = t_i$$

Where the first equality comes from the fact that $C^*v^1 = t^*$, the second comes from the fact that $\sum_{j=k+1}^{n} C_{i,j} \cdot v_j = \oplus_{\{j>k:C_{i,j}=1\}} v_j$ and the last equality is by the definition of t_i^*.

The probability that the sub matrix C^* doesn't have full rank, for a randomly chosen matrix C, is upper bounded by the number of linear row combinations possible ($2^{\frac{n}{4}}$ as there are $\frac{n}{4}$ rows) times the probability that the linear combination will be the all zero vector ($2^{-\frac{n}{3}}$ as each vector has $\frac{n}{3}$ coordinates). So the probability that C^* doesn't have full rank is bounded by $\frac{2^{\frac{n}{4}}}{2^{\frac{n}{3}}} = 2^{-\frac{n}{12}}$, which is negligible. Thus we can find one full rank matrix C which will be used always, so we don't even have to generate a new matrix every time and check it is valid.

We need to verify that the response vector v is uniformly distributed over the space of possible solutions to $Cv = t$, in order to achieve the indistinguishability property we require from the protocol. Denote by V_t the set of solutions for $Cv = t$ and by V_{t,v^2} the set of solutions for $Cv = t$ where we restrict the last $n-k$ bits of v to be v^2. Now we note that for every $v_1^2 \neq v_2^2$ the sets V_{t,v_1^2} and V_{t,v_2^2} are disjoint, as we restrict the last $n-k$ bits of the solutions to be different. Moreover we know that the two sets V_{t,v_1^2} and V_{t,v_2^2} are of equal cardinality, as they are solutions to the same full rank matrix. We draw the bits $v^2 = (v_{k+1}, \ldots, v_n)$ uniformly at random, which means we choose the set V_{t,v^2} uniformly at random and from V_{t,v^2} we draw the solution again uniformly at random. Since all the sets are of equal size, disjoint and of equal likeliness to be picked, we get a uniformly at random chosen vector $v = (v^1, v^2)$, over the solutions to $Cv = t$.

We need to verify that the resulting protocol is indeed complete, sound and indistinguishable like we required. We proved **completeness** when we showed that we can find a suitable response vector for the protocol with probability $(1 - 2^{-\frac{n}{12}})$. Regarding **soundness**, as there is a space of size $2^{\frac{n}{4}}$ of vectors that satisfy $Cv = t$ and a cheating *secondary* can only guess one vector and hope it falls in that space (as it doesn't know any secrets), it has a negligible probability of $2^{-\frac{n}{4}}$ of succeeding in cheating the *resolver* and successfully proving $x \notin S$ for $x \in S$.

To argue **indistinguishability**, the critical point is that we generate a response vector which is distributed uniformly at random over the solutions, thus the modified protocol remains indistinguishable as we

executed the original protocol (which was ZK) and responded with a vector uniformly random over the constraints. A simulator simply generates the parameters for the scheme, i.e. the mapping M and the modulus N. Using the factorization of N it can compute all the square roots to all the quadratic residues. Thus in the protocol itself, after receiving the target vector t it draws uniformly at random a solution to $Cv = t$, which is easy to do as C has full rank (because its sub matrix C^* has full rank), computes b_v and returns v and b_v. The view of this simulation is statistically close to that of a real execution of the protocol, since the public key is generated using the same algorithms used by the *primary* and the protocol itself produces the same distribution on the response vector, i.e. a uniform distribution. Because a real system has the negligible probability to fail (when it can't find a vector to fulfill the constraints), we get that the protocol is only statistically indistinguishable.

Performance: Consider the performance of the resulting denial of existence mechanism. For the Pedersen commitments, the proofs of inequality require the *secondaries* and *resolvers* to do a constant number of modular exponentiations (9 for the *secondary* and 8 for the *resolver*) and a small constant number of multiplications which is negligible with respect to the exponentiations. The fixed set non-membership proofs requires the *resolver* and *secondary* to make at most n modular multiplications and the *secondary* also has to do a Gaussian elimination process to the matrix of size $\frac{n}{4} \times \frac{n}{3}$, which for a binary matrix is again, negligible with respect to the exponentiations of the Pedersen commitments. An exponentiation modulus N is about $\log N$ multiplications, and as $N < n$, we see that the multiplications required for the fixed set non-membership amount to less than one exponentiation per party. Thus the dominating factor of this scheme is the modular exponentiations, and as we need to perform only a small constant number of them this scheme is very efficient.

7 PSR from Unpredictability or VRF, VUF and PRF Based Constructions

We show a few constructions for PSR systems based on variants of *Verifiable Random Functions* (VRFs for short) and *Verifiable Unpredictable Functions* (VUFs for short) and then show a construction that uses *Pseudorandom Functions with interactive ZK proofs* and discuss constructions in the random oracle model.

All constructions in this section follow the same guideline for proving non-membership in their design of a denial of existence mechanism. We use a technique reminiscent of a binary search where in order to prove non-membership, one shows that the queried element falls between the values of a committed set. Use a function F that appears random on a preselected set of elements (R in our case), to compute the values of the function over the set ($F(R) = \{F(x_i)|x_i \in R\} = \{y_i\}$) and sign all the values in pairs, lexicographically. Then in order to prove non-membership for $x \notin R$, the *secondaries* prove the value $F(x)$ in a ZK fashion while also presenting a signature over (y_i, y_{i+1}) where $y_i < F(x) < y_{i+1}$ in order to convince the *resolver* of the non-membership. This type of technique was also used for NSEC and NSEC3 [5,50] in order to define a secure DNS protocol (DNSSEC). The first didn't use any function on the addresses in the domain while the second only used a publicly known hash function on the addresses, thus resulting in a protocol which is susceptible to zone enumeration attacks.

In Section 7.1 we define the primitives VRF and VUF and their variants tsVRF and tsVUF. In Section 7.2 we show the construction which uses tsVRFs and prove its correctness and in Section 7.3 we do the same with the tsVUF based construction. In Section 7.4 we describe the most appropriate VRF and VUF constructions from the literature and consider the complexity of the resulting schemes, when using VRFs and VUFs instead of tsVRF and tsVUFs. In Section 7.5 we show the construction based on the Naor-Reignold PRF [59]. In Section 7.6 we do the same for the random oracle construction based on the famed BLS signature scheme [18] and also discuss other random oracle constructions.

7.1 VRF and VUF Definition

Verifiable random/unpredictable functions, defined by Micali et al. [55], are functions which look random or unpredictable, in the computational sense, but a prover (who knows a secret key) can prove the value of the function on a given element, to a verifier that knows only the public key of the function. In our constructions we use a weaker variant of both primitives. We will define both the original primitives and their variations, as we will also consider implementations of VRFs and VUFs for our constructions (although they have stronger requirements than we need). All the discussed primitives are verifiable functions and use the following three algorithms:

Definition 8. *A verifiable function (VF) is a function* $F : \{0,1\}^{seed(k)} \times \{0,1\}^{in(k)} \rightarrow \{0,1\}^{out(k)}$*, where seed, in and out are polynomials in the security parameter* k *and are efficiently computable.* F *has three algorithms* $(Setup, Prove, Verify)$ *such that:*

$Setup(1^k)$**:** *gets as input the security parameter* k *and outputs the public and secret keys* (PK, SK)*.*
$Prove_{SK}(x)$**:** *gets as input the secret key* SK *and* $x \in \{0,1\}^{in(k)}$ *and outputs the evaluation of the function on* x *and its proof,* $(F_{SK}(x), \pi_{SK}(x))$*.*
$Verify_{PK}(x, y, \pi)$**:** *gets a proof* π *that* $F_{SK}(x) = y$ *and verifies it is correct. Returns* 1 *when the proof is valid and* 0 *otherwise.*

We require three properties from verifiable functions. The completeness requirement, which is called *provability*, states that if the value of the function and its proof over x were computed honestly, then they will be verified successfully with probability 1. More formally:

Definition 9. ***Provability.***
If $Prove_{SK}(x) = (y, \pi)$ *then* $Verify_{PK}(x, y, \pi) = 1$*.*

The second requirement is a soundness condition, which is called *uniqueness* and *trusted uniqueness*. This requirement states that no false statements could be proven in the system, i.e. that $F(x)$ is unique and cannot be proven to be a different value either for all public keys or for validly chosen public keys. We will require not only that our verification accepts at most one evaluation per element, but also that an overwhelming fraction of the domain has an evaluation (i.e. some value which can be verified successfully). More formally:

Definition 10. ***Uniqueness and Trusted Uniqueness.***

1. ***Uniqueness.*** *For every public key* PK *there doesn't exist any tuple* $(x, y_1, y_2, \pi_1, \pi_2)$ *such that* $y_1 \neq y_2$ *and both* $Verify_{PK}(x, y_1, \pi_1) = 1$ *and* $Verify_{PK}(x, y_2, \pi_2) = 1$*. For every such* PK *there is an overwhelming fraction of the domain for which for every* x *there is a* y *and* π *such that* $Verify_{PK}(x, y, \pi) = 1$*.*
2. ***Trusted Uniqueness.*** *For every* ***validly*** *chosen public key* $PK \in Setup(1^k)$*, there doesn't exist any tuple* $(x, y_1, y_2, \pi_1, \pi_2)$ *such that* $y_1 \neq y_2$ *and both* $Verify_{PK}(x, y_1, \pi_1) = 1$ *and* $Verify_{PK}(x, y_2, \pi_2) = 1$*. For every such* PK *there is an overwhelming fraction of the domain for which for every* x *there is a* y *and* π *such that* $Verify_{PK}(x, y, \pi) = 1$*.*

The third condition we require determines the randomness of the function. We can choose between *pseudorandomness* and *unpredictability*, where the first means one cannot distinguish between a random value and $F(x)$ and the latter means one cannot compute $F(x)$. We can also choose between *existential* and *selective* randomness, i.e. whether the adversary in the security game gets to choose its target element x ahead of time (before getting the public key) or at the time of its choice. We define the 4 notions of randomness more formally:

Definition 11. ***Pseudorandomness and Unpredictability both selective and existential.***

1. ***Selective Pseudorandomness.*** *All probabilistic polynomial time adversaries with oracle access to* $Prove_{SK}(\cdot)$ *cannot* ***distinguish*** *between a random value* $r \in \{0,1\}^{out(k)}$ *and* $F_{SK}(x)$ *with more than a negligible advantage, where the adversary gets to choose* x ***ahead of time****, i.e. before getting the public key* PK*. The adversary is not allowed to query the oracle on* x *at any point in time.*

2. ***(Existential) Pseudorandomness.*** *Similar to selective pseudorandomness, but the adversary gets to choose its target x at* ***any time*** *it chooses to.*
3. ***Selective Unpredictability.*** *All probabilistic polynomial time adversaries with oracle access to $Prove_{SK}(\cdot)$ cannot* ***compute*** *$F_{SK}(x)$ with more than negligible probability, where the adversary gets to chooses x* ***ahead of time****, i.e. before getting the public key PK. The adversary is not allowed to query the oracle on x at any point in time.*
4. ***(Existential) Unpredictability.*** *Similar to selective unpredictability, but the adversary gets to choose its target x at* ***any time*** *it chooses to.*

Originally Micali et al. defined VRFs to be verifiable functions with provability, uniqueness and (existential) pseudorandomness and VUFs to be the same but with (existential) unpredictability instead of pseudorandomness. Their requirements are too strong for our needs in constructing PSR systems. Instead of uniqueness we only require from our functions to have trusted uniqueness, as the party which will generate the keys for the verifiable functions will be the *primary*, which is a trusted party, thus we do not need the very stringent requirement of uniqueness to hold for all public keys, just for validly chosen ones. Note that Brakerski et al. [20] presented a different weakening for verifiable random/unpredictable functions, where the pseudorandomness/unpredictability only holds for randomly chosen elements. This primitive is too weak for us, as we are not guaranteed that the set of values $F(R)$ will look random, as R is not chosen randomly, which is critical for the ZK property we desire from our construction.

We show two constructions of PSR systems with verifiable functions, one with pseudorandom functions and the second with unpredictable functions. But again we do not need to use the existential notion of randomness defined originally for VRFs and VUFs, as the selective version will suffice for our constructions. The pseudorandomness property gives us the ability to replace the set of values $F(R) = \{F(x_i)|x_i \in R\}$ with a set of random values, without an adversary noticing the difference, which gives us the zero-knowledge property we desire. As the set R is chosen ahead of time by the *primary*, the notion of selective pseudorandomness will suffice for our needs. We use the unpredictable functions to construct pseudorandom functions, so again selective security will suffice. As most previous work on the subject concentrated on constructing VRFs and VUFs, we can use existing constructions of VUFs and VRFs and plug them into the constructions described in Section 7.2 and in Section 7.3. Besides such existing constructions, we can also use other constructions such as the GHR signature scheme [33], described in Section 7.3, which is not a VUF, as it only has the trusted uniqueness property, but trusted uniqueness suffices for our needs.

We call verifiable functions which have provability, trusted uniqueness and selective pseudorandomness *trusted selective verifiable random functions* (or tsVRF for short) and the selective unpredictable variant will be called *trusted selective verifiable unpredictable functions* (or tsVUF for short).

7.2 Constructing PSR Systems from tsVRFs

We describe how to construct a PSR system which uses tsVRFs and signatures (see definition in Appendix 11.2). We use the following notation for clarity:

- F: Denotes the tsVRF function we use and F_π denotes the proof for the value of F. F maps elements in U to $\{0,1\}^n$, where we choose n to be large enough to avoid birthday attacks, so that $\frac{|R|}{2^n}$ is negligible.
- $Sign$: A signature over a message with the relevant secret key. As for all of our constructions, we use consistent signature schemes, which always produce the same signature on the same message.
- $R = \{x_1, \ldots, x_r\}$: is the privileged set and $V = \{v_1, \ldots, v_r\}$ are the corresponding values.

The *primary* computes the values of F over the set R and sorts them in lexicographical order, $y_1, \ldots, y_r$. We then use the signature scheme to sign every pair of adjacent values, as well as adding an opening and closing values $y_0 = 0^n, y_{r+1} = 1^n$. Denote those signatures $Sign(y_j, y_{j+1})$. We also use the signature scheme in order to sign every element in R. We denote the signatures and their values as $\{s_i\}_{i=1}^r$ where $s_i = (Sign(x_i, v_i), (x_i, v_i))$.

Now when queried on an element $x \notin R$ the *secondary* calculates $F(x)$ and finds $j \in \{0, \ldots, r\}$ for which $y_j < F(x) < y_{j+1}$ and returns $(F(x), F_\pi(x))$ and the signature $Sign(y_j, y_{j+1})$. This way the *resolver* can

verify the signature, the tsVRF value with its proof and that $y_j < F(x) < y_{j+1}$, in order to validate that $x \notin R$. When queried on an element $x \in R$ the *secondary* simply returns the signature s_i corresponding to the queried element x and its value v_x.
The three algorithms for the PSR system are:

$Setup(R, V, 1^k)$**:** This algorithm gets the privileged set R and its corresponding values V along with the security parameter k. It runs the setup algorithm for the tsVRF to obtain (PK_{vf}, SK_{vf}) and the setup algorithm for the signature scheme and obtains (PK_{sig}, SK_{sig}). Define the public key to be $PK = (PK_{vf}, PK_{sig})$.
Now compute F over every $x_i \in R$ and sort them by their lexicographical order: $(y_1, \ldots, y_r)$, where $F(x_i) = y_i$. Add $y_0 = 0^n$ and $y_{r+1} = 1^n$ and for each $j \in \{0, \ldots, r\}$ use the signature scheme to generate the signatures: $Sign(y_j, y_{j+1})$. Use the same signature scheme to calculate for every $x_i \in R$:

$$s_i = (Sign(x_i, v_i), (x_i, v_i))$$

Define the private key to be $SK = (SK_{vf}, \{s_i\}_{i=1}^r, \{Sign(y_j, y_{j+1})\}_{j=0}^r, \{y_j\}_{j=1}^r)$.

$Prove(x, PK, SK)$**:** Gets an element $x \in U$ and the two keys. If there is a signature s_i which corresponds to $x_i = x$ then the *secondary* returns it.
If there is no such signature, then the *secondary* calculates $F(x) = y$ (when possible to compute, else it stops and fails[11]) and $F_\pi(x) = \pi$. Since $\{y_j\}$ is sorted lexicographically, the *secondary* finds an index j for which $y_j < y < y_{j+1}$ and returns $(y, \pi, Sign(y_j, y_{j+1}), y_j, y_{j+1})$.
If it can't find such an index j (there is a collision in the tsVRF), then it fails to prove to the *resolver* its requested query.

$Verify(x, PK)$**:** Gets an element $x \in U$ and the public key PK. It initiates an interactive session with a *secondary*. The *resolver* then sends x and gets one of two possible valid responses. One possible response is (s, x, v) where s is a valid signature on the pair (x, v) and if it verifies s correctly then it accepts and outputs 1. The second possible response is (y, π, t, y_1, y_2) where y is the tsVRF value of x and π is its proof, while t is a signature over the values (y_1, y_2). If the signature t is verified correctly, the tsVRF value is verified correctly and $y_1 < y < y_2$ then it accepts and returns 1, else it returns 0.

Theorem 4. *The three algorithms described above constitute an f-ZK PSR for the function $f(R) = |R|$.*

Proof. In order to show that the above construction constitutes a PSR system we need to prove the following three properties: completeness, soundness and ZK.
Completeness. For all $R \subseteq U$, V and $x \in U$ we need to show that running $Setup(R, V, 1^k)$ in order to obtain (PK, SK) and then running the interactive protocol for the PSR, will result in the *resolver* accepting the *secondary*'s interactive proof with overwhelming probability. In case $x \in R$ then the *primary* generated $s_x = (Sign(x, v_x), (x, v_x))$ and the *secondary* can always prove $x \in R$ by using that precomputed signature s_x, which it gets in the secret key SK.

In the case where $x \notin R$ there could be a problem in proving non-membership. As the trusted uniqueness guarantee assures us that at least an overwhelming fraction of the domain can be evaluated then we know that we fail with a negligible probability due to the fact that there is no $F(x)$. If there exists some $x_i \in R$ for which $F(x_i) = F(x)$ then the *secondary* would not be able to provide a proof for that element. We choose a tsVRF with a large enough range such that $|R|$ is negligible with respect to the size of the range (2^n), which means that the probability for a collision (and failure), i.e. $\frac{|R|}{2^n}$ is negligible. This means that an adversary trying to find a collision with q attempts will succeed with probability $\frac{q \cdot |R|}{2^n}$. But if we think of a dynamic case, like the case of DNSSEC, where the set R keeps changing and an adversary can have an influence over the choice of R, then it could just try and find a collision over two elements in the universe U, put one of those elements in R and the second outside of R, thus finding a collision to violate completeness. This will succeed with probability $\frac{q^2}{2^n}$ by the birthday paradox.

[11] As the guarantee for uniqueness (see Definition 10) only assures that an overwhelming fraction of the domain can be evaluated, there might be some $x \in U$ for which there is no $F(x)$, so we stop and fail.

Note that for an adversary *without* knowledge of the secret key, it is also infeasible to find an element $x \in U$ for which the completeness condition doesn't hold. $x \in R$ can always be proven as we precompute a signature for membership and for $x \notin R$ one must find a collision for x. Without knowledge of the secret key finding a collision with non-negligible probability violates the selective pseudorandomness property of the tsVRF, as the set R is chosen ahead of time. Thus it is infeasible to find $x \in U$ for which the completeness condition doesn't hold. Regarding adversaries *with* knowledge of the secret key, they might be able to find collisions for the tsVRF (because they now hold SK_{vf} which might be helpful in finding collisions) and by that find some $x \notin R$ for which $F(x) \in \{y_j\}_{j=1}^{r}$. They might also be able to find $x \notin R$ for which there is no value y which can be verified as $F(x) = y$, which also causes the *secondary* to fail in its proof attempt.

In the case where the entire domain can be evaluated (i.e. every element has exactly one valid evaluation), then we can augment the PSR system a bit to achieve *perfect* completeness by adding another type of proof for $x \notin R$. If it holds that for $x \notin R$ there is some $x_i \in R$ for which $F(x) = F(x_i)$, then a valid proof for $x \notin R$ is $(F(x), F_\pi(x), x_i, F_\pi(x_i), s_i)$, as we simply prove that x has the same tsVRF value as x_i and $x_i \in R$. It is obvious that although this achieves perfect completeness, it compromises the f-ZK property as we reveal information about an element that was not queried ($x_i \in R$), but as collisions occur with negligible probability this will happen rarely and thus will not violate the f-ZK property, i.e. distinguishing between a simulator and a PSR system would still only happen with at most a negligible advantage.

Soundness. The soundness property follows from the existential unforgeability of the signature scheme we are using. We show that if we have a polynomial time adversary A that can break the soundness property with probability ε, then we can construct an adversary B that runs in similar times to A and breaks the existential unforgeability of the signature scheme with probability ε. In order to do that B needs to win a security game where it gets the public key of a signature scheme, gets to ask an oracle for signatures on messages of its choice and then has to forge a signature it didn't query its oracle for.

B first gets the public key for the signature scheme, PK_{sig} and runs A to get the set R and values V on which A can cheat. Next, B runs a modified version of the setup algorithm for the PSR system in order to give A the public and secret keys for the PSR, (PK, SK). Instead of running the setup algorithm for the signature scheme to generate PK_{sig}, it uses the key it got from the security game and in order to produce the two sets of signatures: $Sign(y_j, y_{j+1})$ and $s_i = (Sign(x_i, v_i), (x_i, v_i))$, B uses the signing oracle to generate the signatures for him. B gives the resulting keys of the PSR (PK, SK) to A which runs and returns an attempt to break the soundness property.

B first checks if the attempt to break the soundness succeeded, by verifying the response from A and validating it is truly a false statement. If A succeeds in its attempt then B can forge a signature. If A returned (s, x, v) for $x \notin R$ where s is a valid signature (x, v) then B returns s as its forgery attempt, as B didn't query on (x, v) because $x \notin R$. If A returned (s, x, v') for $x \in R$ where s is a valid signature on (x, v'), but $v' \neq v$, where v is the real value assigned to x, then B returns s as its forgery attempt, as B didn't query on (x, v') because v' is not x's value. If A returned (y, π, t, y_1, y_2) for $x \in R$, where π is a proof that $F(x) = y$, $y_1 < y < y_2$ and t is a valid signature over (y_1, y_2) then B returns t as its forgery attempt. We know B didn't query on (y_1, y_2) because we signed the couples (y_j, y_{j+1}) lexicographically and one of them is $y = F(x)$, which contradicts the fact that $y_1 < y < y_2$. Also note that by the trusted uniqueness property of tsVRFs there exists only one valid evaluation of every element x (as the keys were generate honestly by B), so no one can provide a *false* value for the tsVRF together with a valid proof for that value.

Therefore, we conclude that B succeeds in its forgery attempt whenever A succeeds in its attempt to break the soundness, which means the soundness property indeed follows from the existential unforgeability property of the signature scheme.

f-ZK. In order to show that for $f(R) = |R|$ this construction is f-ZK (in the computational sense) we need to show a suitable simulator SIM, where no probabilistic polynomial time adversary can distinguish an interaction with the real *secondary* in the system and the simulator SIM.

On its first step of the computation $SIM^R(1^k, 1^{|R|})$ runs the setup algorithm for the tsVRF to obtain (PK_{vf}, SK_{vf}) and also runs the setup algorithm of the signature scheme and obtains (PK_{sig}, SK_{sig}). SIM randomly selects $|R|$ values out of F's range, sorts them lexicographically, $y_1, \ldots, y_r \in \{0,1\}^n$ and generates the signatures $\{Sign(y_j, y_{j+1})\}_{j=0}^{r}$ where we add the end points $y_0 = 0^n$ and $y_{r+1} = 1^n$ as the setup algorithm

originally does. The simulator then outputs $PK^* = (PK_{vf}, PK_{sig})$ and a fake simulator key:

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

which we can see is similar to the original secret key that the *secondary* usually gets but it is missing the signatures $\{s_i\}_{i=1}^{r}$ and has the secret key for the signature scheme instead.

On its next rounds of interaction with the adversary, for each query for an element x_i, SIM uses its oracle access to R to check if $x_i \in R$ or not, if it is then SIM also gets its value v_i. If $x_i \in R$ then SIM generates a new signature $s_{x_i} = (Sign(x_i, v_i), (x_i, v_i))$ and sends s_{x_i} back to the *resolver* as a response. Since the signer produces unique (consistent) signatures on the same query we will always get the same signature on the same message. If $x_i \notin R$ the simulator computes $(F(x_i), F_\pi(x_i)) = (y_{x_i}, \pi_{x_i})$ 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 respond to the *resolver* with $(y_{x_i}, \pi_{x_i}, Sign(y_j, y_{j+1}), (y_j, y_{j+1}))$. If we can't find such a j, i.e. a collision has occurred, we abort as we fail to produce an indistinguishable view. Note that we abort only with negligible probability, as we chose the range of the tsVRF to be large enough so that $\frac{|R|}{2^n}$ is negligible, which is the probability for a collision of a single element with the set R.

Now we need to show that the view of the adversary communicating with the simulator is indistinguishable from 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 $x_i \in R$ are signatures coupled with their values $(Sign(x_i, v_i), (x_i, v_i))$, generated the same way the original proofs are computed 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 the elements of R, it gets random values. However, by the definition of tsVRFs their output is selectively pseudorandom, thus 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 (as the set R is chosen before hand, thus selective pseudorandomness suffices). Thus a distinguisher cannot distinguish the simulation from a real execution, even if it knows R. □

7.3 Constructing PSR Systems from tsVUFs

After seeing how to use tsVRFs in order to construct a PSR system, we consider switching the tsVRFs with tsVUFs, that is, replacing the selective pseudorandomness with selective unpredictability (see Definition 11). tsVUFs are obviously weaker, but this may allow us to use constructions which are more efficient than tsVRFs or based on weaker assumptions. Some VRF constructions [55,53] first construct a VUF and then use hardcore bits to transform that VUF into a VRF, which would be redundant in our case if we can use tsVUFs instead of tsVRFs.

The first question is whether the construction in Section 7.2 remains a PSR system when we switch tsVRFs with tsVUFs. The answer is no, the resulting construction is not necessarily a PSR when constructed using tsVUFs. For example, if we take a tsVRF and instead of just outputting its value on each element we concatenate its value with its proof. We would get a tsVUF, as the function isn't selectively pseudorandom but still selectively unpredictable, but the resulting scheme will not be a PSR as each proof of non-membership reveals two elements in the set R, violating the ZK property.

We can use tsVUFs in order to "extract" a few pseudorandom bits from the unpredictable values and by that construct tsVRFs with a small range. This may be seen as a bottleneck, since to get full fledged VRF with long output we need to apply the VRF many times. However, as we shall see, effectively we will be using it only once. By using unpredictable functions instead of pseudorandom ones, we can get functions which are based on weaker assumptions (e.g. computational assumptions as opposed to decisional ones) and also improve the computational efficiency of the scheme.

Assuming we have a tsVUF $f : \{0,1\}^n \to \{0,1\}^m$, we can construct a tsVRF by outputting a series of hardcore bits for f, for example the Goldreich-Levin hardcore bit [39] with different public random strings $r_1, \ldots, r_\ell \in \{0,1\}^m$ after we XOR x with a random string $r^* \in \{0,1\}^n$ as well. We denote $\boldsymbol{r} = (r_1, \ldots, r_\ell, r^*)$ and let $F_{\boldsymbol{r}} : \{0,1\}^n \to \{0,1\}^\ell$, be the resulting function. More specifically, the i^{th} bit is:

$F_{\boldsymbol{r}}(x)_i = \langle f(x \oplus r^*), r_i \rangle \bmod 2$. Note that it is critical that ℓ is at most logarithmic in n, as Goldreich-Levin showed that it is hard to invert a logarithmic number of hardcore bits. For more on the difficulties of transforming unpredictability into pseudorandomness, see [32], which present black box reductions of one-way permutations to different cryptographic primitives and specifically pseudorandom generators.

Claim. If f is a tsVUF then the function $F_{\boldsymbol{r}}$ is a tsVRF.

Proof. The provability property remains intact for $F_{\boldsymbol{r}}$, as in order to prove $F_{\boldsymbol{r}}(x) = y$ one first proves that $f(x \oplus r^*) = y'$ and then using the public strings $r_1, \ldots, r_\ell$ can verify the value of $F_{\boldsymbol{r}}(x)$. Trusted uniqueness holds since $\boldsymbol{r}$ is public and we already have the trusted uniqueness property for f, thus $F_{\boldsymbol{r}}$ is determined uniquely when its keys are chosen honestly. The main issue of this claim is to prove that the resulting $F_{\boldsymbol{r}}$ is selectively pseudorandom. Naor and Reingold show in their paper [60, Theorem 5.1] that if f is unpredictable, then a sequence of GL hardcore bits is pseudorandom against a random challenge even when the random strings $r_1, \ldots, r_\ell$ are public[12]. Because we XOR each x with a random string r^* before we use the tsVUF and the hardcore bits and the fact that r^* is chosen after the adversary (in a selective pseudorandomness game) chooses its target challenge, its choice is essentially random. Thus $F_{\boldsymbol{r}}$ is selectively pseudorandom, completing the proof that $F_{\boldsymbol{r}}$ is a tsVRF. □

Note that the problem with this newly constructed tsVRF is that it has a small range, as for each additional bit in the function's range we need an extra public string r_i, thus using a large range will damage our efficiency significantly. This is why we cannot "plug-in" this tsVRF construction directly into the construction of Section 7.2.

Remark 4. Micali et al. and Lysyanskaya [55,53] both use the Goldreich-Levin hardcore bits to turn VUFs into VRFs, but they use VUFs with small input spaces (only slightly super-polynomial in the security parameter) and this technique doesn't work for large exponential size input spaces, which is why we construct tsVRFs with small range. Naor and Reingold [60] give a counter example to show that when the random strings are public we don't necessarily get a pseudorandom function by using this technique.

Assuming we have a tsVUF scheme, with domain $\{0,1\}^n$ and range $\{0,1\}^m$ where $m = O(n)$ (as we don't need a larger range, just enough to avoid noticeable probability for collisions), we describe how to modify the tsVRF based PSR system to a tsVUF based PSR system.

Instead of using one tsVRF to prove non-membership we will use c tsVRFs ($F_{\boldsymbol{r}}$ described above) and prove that $x \notin R$ by showing that x doesn't collide with the set R for *one* of the c functions. We choose the range of the tsVUF to be large enough to avoid collisions, but we need to specify two more parameters in the scheme: c the number of functions being used and ℓ the output length of the tsVRF. We set $\ell = \max\{2\log|R|, 2\log n\}$. This means that the probability of an arbitrary $x \in U$ to collide with some $x_i \in R$ for one such $F_{\boldsymbol{r}}$ is at most $\frac{1}{|R|}$ ($\frac{1}{n}$ resp.), by a union bound. Thus the fact that we use c functions means that the probability that an element $x \in U$ collides with *all* c functions on at least one point in R is $\frac{1}{|R|^c}$ ($\frac{1}{n^c}$ resp.), by the independence of the choice of keys for the tsVRFs. Thus we choose $c = \log_{|R|} 2^{2n} = \frac{2n}{\log|R|}$ ($c = \frac{2n}{\log n}$ resp.), which leads to the probability for a collision on all the functions for at least one $x \in U$ to be $\frac{2^n}{2^{2n}} = \frac{1}{2^n}$, again by a union bound. In order to prove non-membership, the *secondary* will choose uniformly at random a function to prove non-membership with, if it fails, it chooses another function at random. This will take the *secondary* only $O(1)$ tries to find a suitable function. We claim that the resulting system is a PSR system.

Proof Sketch. Completeness for elements $x \in R$ still holds as those proofs remain the same and completeness regarding $x \notin R$ holds as we just showed that the probability for a collision of some x with all c

[12] Actually Naor and Reingold prove something stronger: this GL construction transforms functions which are unpredictable against a random challenge into functions which are pseudorandom against a random challenge. This means it suffices to use a tsVUF which is unpredictable against a random challenge, which is a weaker requirement than selective unpredictability required for tsVUFs.

functions on the set R has negligible probability. Soundness is implied again by the trusted uniqueness property of the functions we use, combined with the unforgeability of the signature scheme we use, as one cannot provide false proofs for values of the functions or forge signatures with more than a negligible probability.

In order to prove the f-ZK property we build a simulator the same way we did before (this time it has to draw more random values, $|R| \cdot c$). The transcript of the adversary communicating with a system or a simulator remains the same besides proofs for $x \notin R$. But again as R is being chosen by the *primary* before the adversary starts issuing queries, we can still claim that an adversary won't be able to distinguish between the random values chosen by the simulator and the (selectively) pseudorandom values of the c functions over the set R, used in the scheme. Thus it doesn't matter even if we reveal all c functions' evaluation of x, even if it causes the adversary to witness a collision, since due to the small range of the functions collisions will occur frequently for elements outside of R. So by proving the value for one uniformly at random chosen function, which doesn't collide with the set R, doesn't reveal any information, thus completing the proof that the scheme is a PSR system. □

Notice that in the resulting scheme it takes longer time to execute the setup algorithm and the cryptographic keys are longer, but the membership proofs are just as efficient as in the previous scheme. We need to run the tsVUF setup algorithm for c times and also increase the size of the cryptographic keys because of the extra parameters we add (for the public key) and the signatures over the values of the set R for every tsVUF (for the secret key). Despite that, efficient implementations of tsVUFs result in efficient *non*-membership proofs. Thus we have a trade off between the setup running time and key length, and the non-membership proofs' efficiency.

Remark 5. One may notice that every signature scheme that offers unique signatures (the verification algorithm accepts at most one signature per message under any chosen public key, even an improperly chosen key) which is also selectively secure is also a VUF, as mentioned by Micali et al. [55]. But, we need a little less than that, we can use signature schemes which are selectively secure and accept at most one signature per message only for *validly* chosen public keys (the difference between uniqueness and trusted uniqueness), as the public keys used in the PSR are generated by the *primary*, which we trust. In the next section we describe such a signature scheme.

The GHR Signature Scheme as a tsVUF

Gennaro, Halevi and Rabin introduced a unique secure hash-and-sign signature scheme [33] which we can use as a tsVUF. Their construction is based on the RSA cryptosystem, but instead of encoding the message as the base of the exponent and keeping a public fixed exponent they keep the base fixed and encode the message into the exponent. In order to prove their signature scheme is existentially unforgeable against an adaptive chosen message attack, GHR use the *strong RSA assumption* (see Section 11.4).

To encode the messages they use a *division intractable hashing family*, which they prove exist, again, under the strong RSA hardness assumption. However, the main issue is how complex the implementation for this function is (see [27]).

Definition 12. *Division intractable hashing family. A hashing family H is said to be division intractable if finding a function $h \in H$ and distinct inputs $X_1, \ldots, X_n, Y$ such that $h(Y)$ divides $\prod_{i=1}^{n} h(X_i)$ is computationally infeasible.*

Here is a description of the scheme:

- **Setup.** Pick a safe RSA modulus $n = pq$ ($p = 2p' + 1$ and $q = 2q' + 1$, where p, p', q, q' are primes) and a random element $s \in \mathbb{Z}_n^*$ and set $PK = (n, s)$ while $SK = (p, q)$.
- **Sign.** For a message m, we use the division intractable hash function h to compute $e = h(m)$ and then sign by computing $\sigma \equiv s^{1/e} \bmod n$.
- **Verify.** To verify that σ is a signature over m simply compute $e = h(m)$ and check if $\sigma^e \equiv s \bmod n$.

We prove next that this signature scheme is indeed a tsVUF, as defined in Section 7.1.

Theorem 5. *The GHR signature scheme is a selective trusted verifiable unpredictable function (tsVUF).*

Proof. First note that this scheme does not have (separate) proofs for its values as it is originally a signature scheme, thus the value of the function on an element can be verified without a proof. In order to prove that GHR is a tsVUF we need to show that it has provability, trusted uniqueness and selective unpredictability:
Provability. This signature scheme is unique and one can easily see that every validly signed message m will be verified correctly with probability 1.
Trusted Uniqueness. If one chooses honestly the public and secret keys for this signature scheme (i.e. chooses a good RSA modulus and a good hash function) then we know that at most one signature per message will be verified successfully. We do not have to worry about $h(m)$ dividing $\varphi(n)$, since this implies factoring n (which we assumed to be hard) and hence a *secondary* can always (except with negligible probability) compute $Sign_{SK}(x)$, which is consistent with the trusted uniqueness definition. When the value of the function on an element is well defined, then only one signature will be verified correctly as only for $\sigma \equiv s^{1/e} \bmod n$, it holds that $\sigma^e \equiv s \bmod n$.
(Selective) Unpredictability. As the signature scheme is known to be existentially secure under the strong RSA assumption, we know that no adversary can compute the signature over its choice of a message m, even with the help of a signing oracle (from which it cannot ask for x's signature). Thus this signature scheme achieves (existential) unpredictability, which is stronger than our requirement of selective unpredictability.
□

Although it is proven by Gennaro et al. that a family of division intractable hash functions can be constructed under the strong RSA hardness assumption, the major issue in implementing this scheme is in constructing an efficient implementation for such a family. They offer an inefficient construction for this family of hash functions, as in order to get this property, they force the hash function to output primes by probing values until finding a prime integer to assign to the element being evaluated. We are not familiar with an efficient implementation for such a family of hash functions, but if one can construct an efficient division intractable hash family, then it results in a very attractive PSR system, as all its algorithms require only one modular exponentiation and at most one hash evaluation. In the next section we discuss existing implementations of VUFs and all three candidates [55,53,28] cannot handle a large input space, thus our prime candidate for constructing a PSR system from tsVUF in the standard model (we offer an efficient construction in the random oracle model, see Section 7.6) is the GHR signature scheme.

We note that in the random oracle model (see Appendix 11.1 for details), i.e. using a random oracle which maps elements into integers modulus n instead of a division intractable hashing family, results in a secure signature scheme, since random oracles are division intractable as shown by GHR. Coron and Naccache [27] do provide a subexponential attack on the GHR signature scheme, but also mention that by doubling the size of the random oracle's output the GHR signature scheme remains secure in the random oracle model. Thus in the random oracle model the GHR signature scheme is a good implementation for a tsVUF.

7.4 Good VRFs, VUFs and Their Complexity

After establishing two constructions of PSR systems, one that uses tsVRFs and one that uses tsVUFs (or signature schemes as in Remark 5), we consider implementations of these primitives. As most work on the subject concentrated on constructing VUFs and VRFs, we will concentrate on them, rather than the relaxation of tsVRFs and tsVUFs respectively. We can choose our underlying VRFs/VUFs to get different properties and underlying cryptographic assumptions from the resulting PSR system. Here is a short recap of important VRFs/VUFs.
Micali, Rabin and Vadhan [55] were the first to introduce VUFs and VRFs. They first construct a VUF based on an assumption they call the RSA′ $s(k)$-hardness assumption (defined in Section 11.4), a variant of the original RSA hardness assumption. This assumption is not a good one as $s(k)$ is the size of the input space. In order to turn this VUF into a VRF they use the Goldreich-Levin (GL) hardcore bits [39]. As we noted in Remark 4 this results in a VRF only for small input spaces and is not secure for large domains. They also suggest to increase the size of the input space by a tree like extension which isn't efficient as for

every k additional bits added to the input, the proof gets k times longer and so does the verification process. Their VUF is similar to the Gennaro-Halevi-Rabin signature scheme [33], albeit it can't handle large input spaces as well.

Lysyanskaya [53] constructs VUFs based on groups where the computational Diffie-Hellman problem is considered hard while the decisional Diffie-Hellman problem is easy, i.e. gap Diffie-Hellman groups (defined in Section 11.4). It uses error correcting codes and a tree like structure for verification. For an n bit input the proof consists of n group elements which are nodes on the way from the root of a binary tree to the leaf corresponding to the input. In order to turn the VUF into a VRF the author uses the same transformation used by Micali et al. which is again inefficient, which results again in a VRF which is secure only for a small input space. In this case the suggested VUF also can't handle large input spaces.

The Dodis-Yamploskiy VRF construction [28] is extremely simple in its description. Let $\mathbb{G}$ be a bilinear group of prime order p, g a generator for $\mathbb{G}$ and e a bilinear map (see Definition 16). We choose some random $s \in \mathbb{Z}_p$ and set $PK = g^s$ and $SK = s$. The function over x will be $F(x) = e(g,g)^{\frac{1}{x+SK}}$ and its proof is $\pi(x) = g^{\frac{1}{x+SK}}$. In order to verify that $y = F(x)$ one needs to verify that:(1) $e(g^x \cdot PK, \pi) = e(g,g)$ and (2) $y = e(g,\pi)$.

Its major advantages are the fact that it uses short keys and proofs (one group element) and its computation and verification are efficient (a constant number of exponentiations or pairing computations). The problem with this VRF is its security which relies on the q-decisional Bilinear Diffie Hellman inversion problem (q-DBDHI, defined in Section 11.4), where q is the size of the domain of the VRF. This security degradation occurs since in the reduction to the q-DBDHI assumption, we need to guess the target challenge and thus have success probability which is degraded by the size of the input space. This makes our domain relatively small under reasonable assumptions (only polynomial in the size of the security parameter). In order to extend the domain to $\{0,1\}^*$ the authors offer two approaches: the one offered by Micali et al. and using collision resistant hash functions to hash the input before using the VRF. The first approach is inefficient and the second might lead to collisions and if the probability for a collision is noticeable then we cannot use the VRF as is for the construction of a PSR system.

Dodis and Yamploskiy also offer a VUF construction which is similar to their VRF construction, it is based on the q-Diffie-Hellman Inversion assumption (q-DHI, defined in Section 11.4), where again q is the size of the domain. This VUF is also very efficient in terms of its computation and has short keys and proofs, but again lacks security for large input spaces.

Abdalla, Catalano and Fiore [1] construct VRFs from a class of identity based encryption schemes which they call VRF suitable. They use IBE key encapsulation mechanisms (IB-KEM) to construct VRFs without having to go through VUFs and the inefficient Goldreich-Levin transformation. Their reduction is tight which means there is no security loss when construing VRFs from IB-KEMs. They use this technique in order to construct two VRFs, the first uses the Sakai-Kasahara IB-KEM [71] which results in a family of VRFs, which can be thought of as a generalization of the Dodis-Yamploskiy VRF. The security analysis for this VRF remains the same as for the Dodis-Yamploskiy construction, thus not gaining any improvement on that matter.

The second construction uses a VRF suitable IB-KEM inspired by Lysyanskaya's VRF which they show to be secure under the decisional ℓ-weak Bilinear Diffie-Hellman inversion assumption (ℓ-wBDHI, defined in Section 11.4), where 2^ℓ is the input space. Its great advantage for us is that this construction constitutes a selective VRF with a large input space (unbounded if needed), unlike the previous constructions of VRFs, which only have a slightly super-polynomial size input space.

In a recent paper [2] Abdalla et al. extend their results by showing that the second construction can also be proven fully secure for such exponentially large input spaces. They show it once, by altering the scheme by hashing the identities with admissible hash functions, which can be constructed from collision resistant hash functions and error correcting codes as shown by Boneh and Boyen [14]. They also prove the original scheme fully secure under a different assumption, the ℓ-Decisional Diffie-Hellman Exponent assumption (ℓ-DDHE, defined in Section 11.4).

Hohenberger and Waters [43] construct VRFs with large input spaces. Their construction is similar in structure to the Naor-Reingold PRF [59] and relies on the ℓ-Decisional Diffie Hellman Exponent assumption

(ℓ-DDHE, defined in Section 11.4) where $\ell = O(q(k) \cdot n)$, $\{0,1\}^n$ is the domain and $q(k)$ is the number of queries an adversary is allowed to use to try and break the pseudorandomness of the VRF. Thus we get good security for large input spaces. They also offer to increase the input size with collision resistant hash functions or the tree like extension. The keys are of size linear in n, a value in the VRF's range is only one group element in size and takes a linear number of multiplications, one exponentiation and a pairing computation to compute. The proof of a value is represented by n group elements and takes a linear number of multiplications and exponentiations to compute (can be reduced to the number of ones in the input). Verification takes a linear number of pairing computations as well.

Boneh, Montgomery and Raghunathan [19] use a technique they call augmented cascade to get two constructions of VRFs, one from the Dodis-Yamploskiy construction and the second from the Hohenberger-Waters construction. They use augmented cascade to get the DY construction with input space of size ℓ^n where n is constant and ℓ is a polynomial in the security parameter, under the $n\ell$-BDH assumption (defined in Section 11.4), unlike the original DY construction which uses the ℓ^n-DBDHI assumption (defined in Section 11.4). In turn, this construction makes the proofs and verifications n times longer and is still not equipped to handle large input spaces. Their second construction has input space of $\{0,1\}^m$ for arbitrary m under the $O(m)$-Bilinear Diffie-Hellman assumption ($O(m)$-BDH, defined in Section 11.4) unlike the HW construction which uses the $O(m \cdot Q)$-DDHE assumption (where Q is number of queries used by an adversary), thus constructing a VRF with large input space under a weaker assumption. In turn this construction is less efficient than the HW construction.

Jager [46] managed to construct both a VRF and a VUF which are proven to be secure (for exponential size input spaces) under cryptographic assumptions which are considerably weaker than previously known VRF and VUF constructions. In order to construct these functions Jager describes a general scheme, which uses hash functions to map the elements in the domain $\{0,1\}^n$ into $\{0,1\}^{p(n)}$, where $p(n)$ is a polynomial. Using admissible hash functions results in a VUF and using **balanced** admissible hash functions results in a VRF. The construction of the VRF is proven to be secure under the decisional ℓ-weak Bilinear Diffie-Hellman inversion assumption (ℓ-wBDHI assumption, defined in Section 11.4) for $\ell = \lfloor \frac{\ln(2Q+\frac{Q}{\delta})}{-\ln(1-c)} \rfloor - 1$, where $0 < c < 1$ is a constant, Q is the number of queries (and is thus also at most a polynomial in n) and δ is the advantage of the adversary in solving the ℓ-wBDHI problem. Since Q and $\frac{1}{\delta}$ are polynomials in the security parameter, we get that ℓ is only logarithmic in the security parameter. While this construction is slightly less efficient then the other fully secure VRFs [2,19,43], asymptotically its efficiency has the same order of magnitude. The VUF Jager proposes is proven to be secure under the ℓ-CDH assumption (ℓ-CDH, defined in Section 11.4) for $\ell = \lfloor \frac{\ln(2Q}{c} \rfloor - 1$, where c, Q, δ are as described before. Its efficiency is similar to that of the VRF and unlike other VUF constructions it is the only one to be secure for exponential size input spaces.

VRF conclusions. As explained above, the first three VRFs we mentioned cannot handle large domains (can only handle slightly super-polynomial size domains), which makes them a bad choice to construct a PSR system with. The security of the VRF is critical for the resulting PSR and we should consider VRFs which are also secure for large (exponential) domains. Thus we can consider the following VRFs:

1. HW [43].
2. BMR_2 [19](the second construction by Boneh et al.).
3. ACF_2 [1](the second construction by Abdalla et al.).
4. Jager [46].

We summarize their properties in Figure 5. The table is for a VRF with domain $\{0,1\}^n$, where n is a polynomial in the security parameter k. Computations are comprised of the number of group multiplications (M), group exponentiations (E), pairing computations (P), balanced admissible hash functions (H) and error correcting codes (EC) we employ in the computation. Sizes of keys and proofs are measured by the number of groups elements needed to represent each proof or key, where if the keys' sizes are different we specify the public key first and then the secret key. In all three construction $F(x)$ is one group element in size. In the HW construction the length of the proof and the number of exponentiations done in the verification and proof computing can be reduced from $n+k$ to $ones(x)+k$ where $ones(x)$ is the number of ones in the input x.

Regarding the security of the scheme, one can look in Section 11.4 for a discussion on the assumptions we use. Generally speaking the assumption made by Jager seems the weakest, as they offer both a weak assumption and its dependency on the input size is only poly-logarithmic as opposed to polynomial in the rest of the constructions, which is why it offers the most secure VRF.

name	Complexity assumptions	Setup time	Keys' size	Computing $F(x)$	Computing $F_\pi(x)$	size of $F_\pi(x)$	Verification
1. HW	$O(q(k) \cdot n)$-DDHE	$(n+2)$E	$n+4$ per key	$(n+2)$M + 1E + 1P	$(n+2)$M + $(n+1)$E	$n+1$	$(n+3)$P
2. BMR_2	$6n$-BDH	48E	$48n + 2$ per key	$(48n)$M + 1E + 1P + 1EC	$(48n)$M + 1P + 1EC	$48(n+1)$	$(49n)$M + $(49n)$E + $(49n)$P
3. ACF_2	n-wBDHI or n-DDHE	$2n+2$E	$2n + 3$ $2n+1$	nE + 1P	nE	$n+1$	$(n+1)$P
4. Jager	$poly\log(n)$ -wBDHI	$2p(n)$E	$2p(n) + 2$ $2p(n)$	1H + $p(n)$M + 1E+ 1P	1H + $p(n)$E	$p(n)$	1H + $(p(n)+1)$P

Fig. 5. Secure VRFs summary.

VUF conclusions. As most work regarding VUFs and VRFs concentrated on trying to construct VRFs, we have only four VUF constructions to choose from. Three of them [55,28,53] have only a slightly super-polynomial size input space, which is not good enough for our needs, as we would like to have an exponential size input space. The fourth VUF, suggested by Jager [46], is proven to be secure for exponential size input spaces and can be utilized for constructing PSRs. Its complexity is similar to that of the VRF suggested in that same paper, which is summarized in Figure 5, with the exception that instead of using balanced admissible hash functions, it suffices to use only admissible hash functions.

Besides those VUFs, as mentioned in Remark 5, one can construct a VUF using a *unique* signature scheme which is selectively secure and verifies at most one signature per message for *validly* chosen keys. In Section 7.3 we suggest such an implementation of a tsVUF using the GHR signature scheme [33], which offers an incredibly efficient tsVUF implementation, assuming we can construct an efficient division intractable hash function, which is also secure for large input spaces. Note that in the random oracle model we can use the BLS signature scheme [18] as a tsVUF, as described later in Section 7.6.

7.5 PRFs with Interactive ZK Proofs

An alternative to using tsVRFs is to allow the proof of correctness of the value to be interactive, that is to use a pseudorandom function (PRF) with a zero knowledge interactive proof of value. To construct a PSR system this way, we modify the tsVRF based PSR system to use the PRF instead of the tsVRF and in order to prove $x \notin R$, *secondaries* will simply prove the value of the PRF interactively in a zero knowledge fashion, instead of a one message proof, which is the case for tsVRFs. If we have PRFs, where their values can be proven interactively with the zero knowledge property (a verifier will not learn any additional information on the PRF at different locations then the ones queried) then this modification results in a PSR system.

Besides the ZK property we require from the interactive proof, we want it to be complete and sound (requirements similar to provability and trusted uniqueness), meaning that if $f(x) = y$ then an honest verifier will always be convinced of that by an honest prover and also that no malicious prover can prove $f(x) = y'$ for $y' \neq y$ with more than a negligible probability. With those three properties one can adapt the proof for the tsVRF based PSR system in Section 7.2 to a PRF with ZK interactive proofs based PSR system.

A good PRF with those properties which can be used to construct a PSR system is the Naor-Reingold PRF [59] which is both efficient to compute and has a ZK interactive proof, which is both sound and complete. We describe the construction and then specify its properties and virtues.

Construction 6 *A function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, is defined such that for every n, a key for the function is a tuple $SK = \langle P, Q, g, (a_0, \ldots, a_n)\rangle$, where P is an n bit prime, Q is a prime divisor of $P-1$, g is an element of order Q in $\mathbb{Z}_P^*$ and $a_0, ., , a_n$ is a sequence of $n+1$ elements in $\mathbb{Z}_Q$. For any $x \in \{0,1\}^n$ we define:*

$$f_{SK}(x) = (g^{a_0})^{\prod_{x_i=1} a_i}$$

This function is proven to be pseudorandom under the Decisional Diffie-Hellman assumption (DDH, defined in Section 11.4) and is considered efficient, as it takes up to n multiplications and two exponentiations to compute the function. Naor and Reingold also offer a ZK protocol for proving $f(x) = y$, which be can used for the construction of the PSR system, where the prover has to perform $O(n)$ multiplications and the verifier has to perform $O(n)$ exponentiations in order to complete such an interactive proof. They also offer an interactive protocol for proving $f(x) \neq y$, which can be used to construct a PSR system as well (simply prove $f(x) \neq f(x_i)$ for every $x_i \in R$), but is less efficient then the proof for $f(x) = y$. If one finds a PRF where proving $f(x) \neq y_1, \ldots, , y_r$ is efficient in a ZK fashion then this could result in an efficient PSR system as well.
Other variations on this function are described in Boneh et al. [19].

7.6 Random Oracle Constructions

In this section we consider a PSR construction analyzed in the random oracle model (see Appendix 11.1 for details). Our candidate function to be the building block for this construction is the famed signature scheme by Boneh, Lynn and Shacham [18] which is known to have short and *unique* signatures.

This signature scheme uses a Gap Diffie-Hellman group $\mathbb{G}$ (GDH, defined in Section 11.4) with a generator g, which is a group where the decisional Diffie-Hellman problem is considered "easy", i.e. there is an efficient algorithm that for $a, b, c \in \mathbb{Z}_p^*$, given (g, g^a, g^b, g^c) decides whether $c = ab$, and the computational Diffie-Hellman problem is considered "hard", i.e. there is *no* efficient algorithm for $a, b \in \mathbb{Z}_p^*$ that computes g^{ab} with non-negligible probability. BLS also uses a full-domain hash function $h : \{0,1\}^* \to \mathbb{G}^*$ which is modeled as a random oracle. Both h and $(\mathbb{G}^*, g)$ are system parameters. Here is a description of the scheme:

Key generation. Pick at random $SK \in \mathbb{Z}_p^*$ and compute $PK = g^{SK}$.
Signing The signature on a message x is $\sigma = h(x)^{SK}$
Verification In order to verify that σ is a valid signature over x one checks that $(g, PK, h(x), \sigma)$ is a valid Diffie-Hellman tuple.

The BLS signature scheme is proven to be existentially secure (see Definition 14) in the random oracle model. Since this signature scheme produces *unique signatures*, it can act as a tsVUF, as it generates a unique value for each element in the domain. To be a suitable implementation of tsVUF we need the scheme to have *provability*, which it has, as validly signed messages will always be verified correctly due to the completeness property of the scheme and *trusted uniqueness* is implied by the uniqueness property of the scheme, as only one signature per message will be verified successfully.

The fact that this signature scheme is existentially unforgeable gives us (existential) unpredictability, since if an adversary can predict a signature over x then it means it can forge a signature, thus violating security. Therefore we have a tsVUF which can be used to construct a PSR as specified in Section 7.3. Furthermore, by applying another hash function to the scheme's output signature, again, modeled as a random oracle, one can get a tsVRF and obtain a PSR system as suggested in Section 7.2.

By the simplicity of the scheme's description, its clear that its algorithms are rather efficient. In order to generate the parameters for c signature schemes (as is the case in the tsVUF construction), one only needs to draw c random values and do c exponentiations, while for the tsVRF construction it only has to

do it once. The *secondary* which computes the signature on the queried element when needed, only has to do one or two hashing computations and one exponentiation per function it computes (as we mentioned, in the tsVUF construction the *secondary* might need to evaluate a few functions until it finds one which has no collisions for the queried element). The *resolver* needs to compute one or two hash functions and employ the decisional Diffie-Hellman algorithm in order to verify a signature, which has quite efficient implementations in properly chosen GDH groups. All and all, we see that this construction is very efficient, especially for the *secondaries* which are the party with the smallest load of work out of the 3 parties in this construction. Another advantage to this scheme is the communication complexity, which is very low, due to the short representation of signatures in a BLS scheme.

In our companion paper [36], another PSR construction in the random oracle model was suggested. It uses an RSA-based key hashing scheme (where the hash function is modeled as a random oracle), which functions as a tsVRF and is then used to construct a PSR system which is similar to the construction suggested in Section 7.2. As we mentioned before in Section 7.3, the GHR signature scheme [33] is a secure tsVUF in the random oracle model and can be quite an efficient implementation for a PSR, as its algorithms only require one exponentiation and a hash computation to sign and verify values.

Programmability issues. Ideally we would like to use random oracles which are non-programmable, i.e. functions which are only modeled to be random but neither party can affect the value of the function at any location, unlike programmable random oracles which can be set at some of its locations by the prover of the scheme. Unfortunately, all the mentioned random oracle constructions we suggest need programmable random oracles. In order to prove the BLS and GHR signature schemes secure (even only selectively) one has to use a programmable random oracle, thus both BLS constructions (the tsVUF and tsVRF constructions) and the GHR construction require programmable random oracles. Regarding the NSEC5 construction we suggested in our companion paper, it also requires a programmable random oracle in order to prove its security.

Improving the verification process. The fact that the verification process for the BLS signature scheme requires verifying that a tuple is a Diffie-Hellman tuple usually means that pairings are required in order to verify the validity of a signature. It also means that the group $\mathbb{G}$ which acts as the function's domain has to be a bilinear group, so we could use a pairing to verify if a tuple is indeed a DH tuple, which forces us to work with limited groups such as groups on supersingular elliptic curves or hyperelliptic curves over a finite field. An added advantage of using groups with no "special" requirements is that it will be easier in practice to map elements in the universe U into that said group.

In order to prove that $\sigma = h(x)^{SK}$ was computed correctly, it is enough to show that the discrete logarithm of $PK = g^{SK}$ to the base of g ($\log_g PK = SK$) is equal to the discrete logarithm of σ to the base of $h(x)$, as it proves that $\sigma = h(x)^{SK}$. Since in the random oracle model we can transform interactive proofs into non-interactive ones, we can use the ZK interactive proof of equality for discrete logarithms based on the adaptation of Schnorr's identification protocol [72] (the proof used in the cuckoo hash based PSR, in Section 6.3), to generate a non-interactive ZK (NIZK) proof for the signature at hand.

If we have 4 group elements $a, b, x, y \in \mathbb{G}$ (where $\mathbb{G}$ is of prime order p), for which we would like to prove that $\log_a b = \log_x y$ we do the following. Define a random oracle $f : \mathbb{G}^6 \to \{0,1\}^m$, where m is chosen to be large enough so that the probability for a collision is negligible. The prover selects uniformly at random $t \in \mathbb{Z}_p$ and computes $w = f(a, b, x, y, a^t, x^t)$ and $s = t + w \cdot SK \bmod p$. The proof for the correctness of the signature will be $\pi = (w, s)$ and will be verified by checking that $w = f(a, b, x, y, a^s b^{-w}, x^s y^{-w})$.

8 Towards Dynamic Solutions

This paper focuses on the static case of PSR systems where the sets R and V are fixed and not updated throughout the lifetime of the data structure. Clearly the dynamic case, where the sets get updated with time, is relevant for some applications such as for DNSSEC, our motivation for defining PSR systems. Hence we provide a short review on the matter with regards to the different solutions suggested in this paper.

First when we talk about the dynamic case, we must consider carefully our requirements for soundness, completeness and zero-knowledge. Are *secondaries* allowed to provide "old" answers? (i.e. proving $x \in R$ after an update that removed x from R.) Who and how is the *primary* allowed to update? (only *secondaries* or also the *resolvers*.) Another issue is zero-knowledge: should the fact that some element was recently added or deleted be secret? For full zero-knowledge the simulator should be limited to asking the set oracle only queries asked by the adversary *at exactly the same time*.

We say that a PSR system is weakly dynamic, when a *secondary* is considered sound if the answer it provides is consistent with any point in time in the evolution of the set, i.e. it may prove "old" statements (but for completeness we require that it will be able to prove the most up-to-date version). If we want to require that the answer received represents the current version of the set (the strong dynamic case), then we either need the *primary* to update the *resolvers* directly (which may be unwieldy), or that the *secondary* provides a proof that the answer is updated to within some set time limit Δ. This requires the *primary* to send some information to the *secondaries* every Δ time units.

For the three general techniques suggested in this paper (Sections 4, 6 and 7) we will discuss how to handle updates either for the weak dynamic case or for the strong dynamic case.

A useful tool for this kind of updates will be *certificate revocation schemes*, as the Naor-Nissim scheme [58], where a trusted party updates other parties about the validity of signatures. This way when the *primary* would like to revoke the validity of some signature (like the case where x is removed from R and we don't want *secondaries* to be able to use a precomputed signature of $Sign(x, v)$), it can use such a scheme. This means that when a signature is used it should be appended with a proof that it has not been revoked. Note that in general, such data structures are not zero-knowledge.

Thus when we don't require ZK with respect to updates, the *primary* can send the *secondaries* a list of the serial numbers of all revoked signatures, which will be delivered to *resolvers* with every proof which uses signatures. The *resolvers* need to verify that they got the most updated list and that the signature being used is not on that list. When we do require ZK we can use a new fresh PSR system for the elements which changed their places since the beginning of the process. If proofs of non-membership are comprised of signatures then we can simply use one PSR with the set containing all elements that switched their membership status. If non-membership proofs do not use signatures we can use two PSR systems, one revoking validity for proofs of membership and one revoking proofs of non-membership[13]. This way when proving membership, a *secondary* will have to add a proof that x is not in the update PSR for membership and when proving non-membership it will have to prove it is not in the update PSR for non-membership. The public keys can either be distributed directly to *resolvers* or given to them (signed by the *primary*) by the *secondaries*.

The HIBE based construction (Section 4) can handle the weak dynamic case well: the *primary* has to generate new signatures for the elements that were added to R and for the ones that were removed from R it should compute a secret key for the HIBE, corresponding to that element. This way *secondaries* would be able to prove both claims (membership and non-membership) for elements which were both in and out of the set R. Handling the strong dynamic case seems harder, as although we can revoke the validity of signatures which are used to prove membership using revocation schemes, revoking the validity of HIBE keys used to prove non-membership seems harder. A possible way of doing it, is creating a new PSR system for the elements that changed their membership and generate a fresh tree of HIBE keys for every Δ units of time. Like we claimed before this means that for every proof, *secondaries* now have to add a proof that the element's membership status hasn't changed. Depending on the changes in the set R, this could require a lot of work[14].

The cuckoo hash based construction (Section 6), cannot handle the weak dynamic case very well. The problem with the weak dynamic case is that when we add a new element to the set, some other elements might move around in the cuckoo hash, even though they were always in the set R. Thus *secondaries* will

[13] Alternatively we can simply add one bit to the universe of elements which determines if we are talking about membership proofs (1) or non-membership (0), thus if $0x$ is in the set then it means x was a non-member and now it is a member, and if $1x$ is in the set then x was a member but was removed.

[14] If the updates to the set only remove elements, then we can handle the strong dynamic case by revoking the validity of the signatures that prove membership and issuing new HIBE keys corresponding to the removed elements.

receive updated signatures for the places in the cuckoo hash that were changed. For example if $x \in R$ was moved from location $i = F_1(x)$ of table T_1 to location $j = F_2(x)$ in table T_2, then *secondaries* now hold two signatures for commitments in those two locations. *Secondaries* can prove x is not in location i in table T_1 by using the updated signature for this location (as x was moved from this spot) and prove it is not in location j of table T_2 by using the outdated signature (as before the update x wasn't placed in that spot). Thus *secondaries* will be able to prove a statement which was always false[15]. This means that either the *primary* has to update the *resolvers* about the revoked signatures or update *secondaries* every Δ time units, which will then prove to *resolvers* that the signatures they are using are indeed still valid. Either way this puts us in the strong dynamic case scenario, where we can update the validity of signatures and by that update all the locations in the cuckoo hash tables that have changed during the update. As we use a method for proving non-membership in a **fixed** set in order to prove that elements are not in the stash, it seems like we have to generate new parameters for this scheme every update that changes the stash, as this set is no longer fixed. Luckily the stash is usually very small (larger than s with probability $O(r^{-s})$) so we would have to update a relatively small portion of the denial-of-existence scheme, as opposed to generate all the parameters from scratch. Proving validity for the signatures can be done using a fresh PSR like we explained before, or with a list of revoked signatures if we do not care about the ZK property of updates.

The constructions based on random/unpredictable functions (Section 7) require updating signatures. Thus they can handle both the strong and weak dynamic cases quite well. In both cases the *primary* updates the *secondaries* with new signatures, by giving them proofs of membership for newly added elements ($Sign(x, v)$) and giving new signatures for proofs of non-membership by updating the intervals (y_i, y_{i+1}) and their signatures. For the weak dynamic case it will allow *secondaries* to prove both membership and non-membership for elements that changed their place from within the set to outside it and vice versa. The rest of the elements in the universe will stay as is (this scheme does not have perfect completeness so for a negligible fraction of the elements outside the set we cannot prove non-membership). For the strong dynamic case the *primary* can do the same and revoke the validity of signatures which should not be used anymore and by that force *secondaries* to only prove statements which are up to date with the most recent update. Again we can choose if to use a fresh PSR to prove validity of signatures or provide a list of revoked signatures, depending if we care about ZK or not. Note that the NSEC5 protocol, suggested in our companion paper [36] as a solution for the DNSSEC zone enumeration problem, is of this type which makes the NSEC5 protocol easy to update.

9 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) 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

[15] Note that we can choose not to update the *secondaries* at all and they will only prove statements regarding the original set R before the updates, thus satisfying the soundness requirement of the weak dynamic case. But we do wish to give honest *secondaries* the ability to prove the most updated statements (in order to satisfy the completeness requirement), without allowing malicious *secondaries* the ability to prove statements which were bogus at any point in time.

modular exponentiations for their computations. It is based on the $O(\log |U|)$-weak decisional Bilinear Diffie-Hellman Inversion assumption[16] (ℓ-wBDHI, defined in Section 11.4). 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.

Our cuckoo hash based PSR construction (Section 6) offers both an appealing technique and an efficient implementation, based on very solid and well studied cryptographic assumptions: factoring and the discrete logarithm (see Sections 11.4 and 11.4 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 (Section 6.3) 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 7) reveals the size of the set R, but has the potential of being very efficient if we can construct a VRF/VUF (or a tsVRF/tsVUF) 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 ([28] 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 [43,19,2,46] are not a lot less efficient than the HIBE construction, as can be seen in Figure 5. If we can implement an efficient division intractable hashing family then the GHR construction (Section 7.3) 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 based PSR (Section 7.6) and the NSEC5 construction described in our companion paper [36] are very efficient, while the first relies on a gap Diffie-Hellman group (see Section 11.4) and the latter on the RSA hardness assumption (see Section 11.4).

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.

10 Acknowledgments

We thank our co-authors from [36], 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 from identity-based key encapsulation. In: EUROCRYPT 2009. pp. 554–571. Springer (2009)
2. 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)
3. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: EUROCRYPT 2010. pp. 553–572. Springer (2010)

[16] Boneh et al. prove this assumption holds in the generic group model [73]. 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.

4. 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)
5. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Resource Records for the DNS Security Extensions. RFC 4034, Internet Engineering Task Force (Mar 2005), http://www.rfc-editor.org/rfc/rfc4034.txt
6. Aumüller, M., Dietzfelbinger, M., Woelfel, P.: Explicit and efficient hash families suffice for cuckoo hashing with a stash. Algorithmica 70(3), 428–456 (2014)
7. Bau, J., Mitchell, J.C.: A security evaluation of DNSSEC with NSEC3. In: NDSS 2010. The Internet Society (2010)
8. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS '93. pp. 62–73. ACM (1993)
9. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: EUROCRYPT '94. pp. 92–111. Springer (1994)
10. Bellare, M., Rogaway, P.: The exact security of digital signatures - how to sign with RSA and rabin. In: EUROCRYPT '96. pp. 399–416. Springer (1996)
11. 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)
12. Berman, I., Haitner, I., Komargodski, I., Naor, M.: Hardness preserving reductions via cuckoo hashing. In: TCC 2013. pp. 40–59 (2013)
13. Boneh, D.: The decision diffie-hellman problem. In: Algorithmic Number Theory, ANTS-III 1998. pp. 48–63. Springer (1998)
14. Boneh, D., Boyen, X.: Secure identity based encryption without random oracles. In: CRYPTO 2004. pp. 443–459. Springer (2004)
15. Boneh, D., Boyen, X.: Efficient selective identity-based encryption without random oracles. J. Cryptology 24(4), 659–693 (2011)
16. Boneh, D., Boyen, X., Goh, E.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT 2005. pp. 440–456. Springer (2005)
17. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. SIAM J. Comput. 32(3), 586–615 (2003)
18. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptology 17(4), 297–319 (2004)
19. 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)
20. Brakerski, Z., Goldwasser, S., Rothblum, G.N., Vaikuntanathan, V.: Weak verifiable random functions. In: TCC 2009. pp. 558–576. Springer (2009), http://dx.doi.org/10.1007/978-3-642-00457-5_33
21. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: CRYPTO 2003. pp. 126–144. Springer (2003)
22. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. IACR Cryptology ePrint Archive 2000, 67 (2000)
23. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (2004)
24. Catalano, D., Fiore, D., Messina, M.: Zero-knowledge sets with short proofs. In: EUROCRYPT 2008. pp. 433–450. Springer (2008)
25. 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)
26. Coron, J.: On the exact security of full domain hash. In: CRYPTO 2000. pp. 229–235. Springer (2000)
27. Coron, J., Naccache, D.: Security analysis of the gennaro-halevi-rabin signature scheme. In: EUROCRYPT 2000. pp. 91–101. Springer (2000)
28. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: PKC 2005. pp. 416–431. Springer (2005)
29. Dutta, R., Barua, R., Sarkar, P.: Pairing-based cryptographic protocols : A survey. IACR Cryptology ePrint Archive 2004, 64 (2004)
30. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: ACM 1998. pp. 409–418. ACM (1998)
31. Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. J. Cryptology 1(2), 77–94 (1988)
32. Gennaro, R., Gertner, Y., Katz, J., Trevisan, L.: Bounds on the efficiency of generic cryptographic constructions. SIAM J. Comput. 35(1), 217–246 (2005)
33. Gennaro, R., Halevi, S., Rabin, T.: Secure hash-and-sign signatures without the random oracle. In: EUROCRYPT '99. pp. 123–139. Springer (1999)
34. Gentry, C., Silverberg, A.: Hierarchical id-based cryptography. In: ASIACRYPT 2002. pp. 548–566. Springer (2002)

35. Ghosh, E., Ohrimenko, O., Tamassia, R.: Verifiable member and order queries on a list in zero-knowledge. IACR Cryptology ePrint Archive 2014, 632 (2014)
36. 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)
37. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
38. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM 33(4), 792–807 (1986)
39. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: ACM 1989. pp. 25–32. ACM (1989)
40. Goldwasser, S., Sipser, M.: Private coins versus public coins in interactive proof systems. In: ACM 1986. pp. 59–68. ACM (1986)
41. Goodrich, M.T., Mitzenmacher, M.: Privacy-preserving access of outsourced data via oblivious RAM simulation. In: Automata, Languages and Programming - 38th International Colloquium, ICALP 2011. pp. 576–587. Springer (2011)
42. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association 58(301), 13–30 (1963)
43. Hohenberger, S., Waters, B.: Constructing verifiable random functions with large input spaces. In: EUROCRYPT 2010. pp. 656–672. Springer (2010)
44. Impagliazzo, R.: Pseudo-random generators for cryptography and for randomized algorithms. Ph.D. thesis, University of California, Berkeley (1990)
45. 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)
46. Jager, T.: Verifiable random functions from weaker assumptions. IACR Cryptology ePrint Archive 2014, 799 (2014)
47. Joux, A.: A one round protocol for tripartite diffie-hellman. In: Algorithmic Number Theory, ANTS-IV, 2000. pp. 385–394. Springer (2000)
48. Joux, A., Nguyen, K.: Separating decision diffie-hellman from computational diffie-hellman in cryptographic groups. J. Cryptology 16(4), 239–247 (2003)
49. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. SIAM J. Comput. 39(4), 1543–1561 (2009)
50. Laurie, B., Sisson, G., Arends, R., Blacka, D.: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155, Internet Engineering Task Force (Mar 2008), http://www.rfc-editor.org/rfc/rfc5155.txt
51. Lewko, A.B., Waters, B.: Why proving HIBE systems secure is difficult. In: EUROCRYPT 2014. pp. 58–76. Springer (2014)
52. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: TCC 2010. pp. 499–517. Springer (2010)
53. Lysyanskaya, A.: Unique signatures and verifiable random functions from the DH-DDH separation. In: CRYPTO 2002. pp. 597–612. Springer (2002)
54. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: FOCS 2003. pp. 80–91. IEEE Computer Society (2003)
55. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: FOCS '99. pp. 120–130. IEEE Computer Society (1999)
56. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: CRYPTO 2001. pp. 41–62. Springer (2001)
57. Naor, M.: On cryptographic assumptions and challenges. In: CRYPTO 2003. pp. 96–109. Springer (2003)
58. Naor, M., Nissim, K.: Certificate revocation and certificate update. IEEE Journal on Selected Areas in Communications 18(4), 561–570 (2000)
59. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: FOCS '97. pp. 458–467. IEEE Computer Society (1997)
60. Naor, M., Reingold, O.: From unpredictability to indistinguishability: A simple construction of pseudo-random functions from macs (extended abstract). In: CRYPTO '98. pp. 267–282. Springer (1998)
61. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: ACM 1989. pp. 33–43. ACM (1989)
62. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: CRYPTO 2002. pp. 111–126. Springer (2002)

63. Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. IACR Cryptology ePrint Archive 2004, 170 (2004)
64. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: Algorithms - ESA 2001. pp. 121–133. Springer (2001)
65. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '91. pp. 129–140. Springer (1991)
66. Pinkas, B., Reinman, T.: Oblivious RAM revisited. In: CRYPTO 2010. pp. 502–519. Springer (2010)
67. Prabhakaran, M., Xue, R.: Statistically hiding sets. In: Topics in Cryptology - CT-RSA 2009. pp. 100–116. Springer (2009)
68. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: ACM 1990. pp. 387–394. ACM (1990)
69. Rosen, A.: Concurrent Zero-Knowledge - With Additional Background by Oded Goldreich. Information Security and Cryptography, Springer (2006)
70. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC 2014. pp. 475–484. ACM (2014)
71. Sakai, R., Kasahara, M.: ID based cryptosystems with pairing on elliptic curve. IACR Cryptology ePrint Archive 2003, 54 (2003)
72. Schnorr, C.: Efficient identification and signatures for smart cards. In: CRYPTO '89. pp. 239–252. Springer (1989)
73. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: EUROCRYPT '97. pp. 256–266. Springer (1997)
74. Shoup, V.: OAEP reconsidered. In: CRYPTO 2001. pp. 239–259. Springer (2001)
75. 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)

11 Appendix

11.1 The Random Oracle Model

As some of our constructions are 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 such as [8,9,10,26,74]. We define the model as in Canetti, Goldreich and Halevi [23]. 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 [62]). Values can be set only the first time someone wishes to know $O(x)$ as the oracle must remain consistent.

11.2 Signature Schemes

We use signatures in our constructions, for that end we define signature schemes and their properties. We define public key signature schemes as in Goldreich [37].

Definition 13. *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 consistent 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 a description of a pseudorandom function to provide the randomness needed to sign m (see [38]).

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 14. *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 $m_1, \ldots, m_q$ to the challenger and gets in return a signature on each of those messages $s_1, \ldots, s_q$ 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.

We also use a signature scheme which guarantees security only against an adversary who witnesses only k signatures before its forgery attempt, thus we define a k time existentially unforgeable signature scheme:

Definition 15. *A signature scheme is k-time existentially secure against chosen message attacks if every probabilistic polynomial time adversary A wins the k existential security, which is identical to the existential security game, but A is allowed to make at most k queries to its signing oracle.*

11.3 Pairing Based Cryptosystems

Some of our constructions of PSR systems use *hierarchical identity based encryptions/signatures* or *verifiable random/unpredictable functions*. Both of these primitives are often implemented using bilinear maps as explained below. We define *cryptographic bilinear maps*:
Let $\mathbb{G}, \mathbb{G}_1$ be two (multiplicative) groups of the same prime order q. Let g be a generator for $\mathbb{G}$.

Definition 16. *An admissible bilinear map (also called a pairing) is a function $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ that satisfies the following three properties:*

1. ***Bilinear.*** *For all $g_1, g_2 \in \mathbb{G}$ and for all $x, y \in \mathbb{Z}$ it holds that $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.*
2. ***Non-degenerate.*** *$e(g, g) \neq 1$.*
3. ***Computable.*** *There is an efficient algorithm to compute $e(g_1, g_2)$ for all $g_1, g_2 \in \mathbb{G}$.*

A group $\mathbb{G}$ is said to be bilinear if the group action in $\mathbb{G}$ is efficiently computable and there exists a group $\mathbb{G}_1$ such there is an admissible bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$.

Pairing based cryptosystems were introduced by Joux for tripartite key exchange [47] and were then used by Boneh and Franklin [17] in order to construct identity based encryption. Since then, pairings have been found useful in many applications such as: short signatures [18,14], identity based encryption/signatures [34,17,16,15], verifiable random functions [53,28,43] and many more constructions and applications. For a more extensive coverage of pairing based cryptosystems see the pairing-based cryptographic protocols survey by Dutta, Barua and Sarkar [29].

11.4 Assumptions

As all of our constructions rely on different computational and decisional assumptions we define the assumptions mentioned in this paper for completeness and discuss which assumptions are strong and which are weak. An important notion with regards to assumptions is that of *falsifiability* defined by Naor [57]. We would like to use assumptions which are falsifiable, meaning if the assumption is wrong than there is a constructive way of showing it. Specifically we will be interested in the complexity of checking the refutation of the assumption which influences how acceptable the assumption is. Thus when considering using a construction, one should verify that the assumptions used for the construction are falsifiable.

Factoring

For a security parameter k, let there be a composite integer n of k bits. In the Factoring problem the adversary is given the tuple n and it must compute its factorization $n = p_1^{e_1} \dots p_m^{e_m}$, where p_i are distinct primes and $e_i > 0$. Formally we define:

$$Suc_A^{factor}(k) = |Pr[A(n) = (p_1, \dots, p_m, e_1, \dots, e_m) \text{ and } n = p_1^{e_1} \dots p_m^{e_m}]|$$

Definition 17. *(Factoring assumption). We say that the factoring assumption holds if for any probabilistic polynomial time adversary A, its success in computing the factorization of a composite k bit integer, $Suc_A^{RSA}(k)$ is negligible in k.*

RSA Hardness Assumption

For a security parameter k, let there be a randomly chosen RSA modulus $n = pq$ (where p and q are of length k), a random exponent $e \in \mathbb{Z}_n^*$ and a random integer $y \in \mathbb{Z}_n$. In the RSA hardness problem the adversary is given the tuple (n, e, y) and it must compute x such that $x^e \equiv y \bmod n$. Formally we define:

$$Suc_A^{RSA}(k) = |Pr[A(n, e, y) = x \text{ and } x^e \equiv y \bmod n]|$$

Definition 18. *(RSA hardness assumption). We say that the RSA hardness assumption holds if for any probabilistic polynomial time adversary A, its success in computing an RSA inverse, $Suc_A^{RSA}(k)$, is negligible in k.*

Strong RSA Hardness Assumption

For a security parameter k, let there be a randomly chosen RSA modulus $n = pq$ (where p and q are of length k) and a random integer $y \in \mathbb{Z}_n$. In the strong RSA hardness problem the adversary is given the tuple (n, y) and it must compute (x, e) such that $x^e \equiv y \bmod n$. Formally we define:

$$Suc_A^{sRSA}(k) = |Pr[A(n, y) = (x, e) \text{ and } x^e \equiv y \bmod n]|$$

Definition 19. *(Strong RSA hardness assumption). We say that the strong RSA hardness assumption holds if for any probabilistic polynomial time adversary A, its success in computing a strong RSA inverse, $Suc_A^{sRSA}(k)$, is negligible in k.*

RSA′ $s(k)$-Hardness Assumption

Let $PRIMES_k$ denote the set of k-bit primes and RSA_k denote the set of composite integers that are the product of two primes of length $\lfloor \frac{k-1}{2} \rfloor$. Let A be any probabilistic algorithm which runs in time $s(k)$ on the input 1^k. A tries to win the following game ($s(k)$-RSA′ hardness game):

1. Select $m \xleftarrow{R} RSA_k$; $\quad x \xleftarrow{R} \mathbb{Z}_m^*$; $\quad p \xleftarrow{R} PRIMES_{k+1}$.
2. Let $y \xleftarrow{R} A(1^k, m, x, p)$.
3. A wins the game if $y^p \equiv x \bmod m$.

Definition 20. *(RSA′ $s(k)$-Hardness Assumption). We say that the RSA′ $s(k)$-Hardness Assumption holds if every probabilistic algorithm which runs in time $s(k)$ on the input 1^k wins the $s(k)$-RSA hardness game with probability at most $\frac{1}{s(k)}$.*

Note that unlike the strong and regular RSA hardness assumptions, this assumption is not a standard assumption in complexity. Micali et al. [55] use it in their construction of VRFs, while using $s(k)$ which is exponential in k making it unfalsifiable (as far as we know).

Discrete Logarithm Assumption

Let $\mathbb{G}$ be a group of prime order p and let g be a random generator for $\mathbb{G}$. In the DL problem the adversary is given h and needs to compute x such that $g^x = h$. Formally we define:

$$Suc_A^{DLP}(k) = |Pr[A(g, h) = x : g^x = h]$$

Definition 21. *(DL). We say that the DL assumption holds in the group $\mathbb{G}$ if for any probabilistic polynomial time adversary A, its success in computing x after seeing the tuple (g, h), i.e. $Suc_A^{DL}(k)$, is negligible in k.*

Computational Diffie-Hellman Assumption

Let $\mathbb{G}$ be a group of prime order p and let g be a random generator for $\mathbb{G}$. In the CDH problem the adversary is given the tuple (g, g^a, g^b) and it needs to compute g^{ab}. Formally we define:

$$Suc_A^{CDH}(k) = Pr[A(g, g^a, g^b) = g^{ab}]$$

Definition 22. *(CDH). We say that the CDH assumption holds in the group $\mathbb{G}$ if for any probabilistic polynomial time adversary A, its success in computing g^{ab}, i.e. $Suc_A^{CDH}(k)$, is negligible in k.*

ℓ-Computational Diffie-Hellman Assumption

Let $\mathbb{G}, \mathbb{G}_1$ be two groups of prime order p equipped with the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ and let g and h be two random generators for $\mathbb{G}$. In the ℓ-CDH problem the adversary is given the tuple $(h, g, g^a, g^{a^2}, \ldots, g^{a^\ell})$ and needs to compute $e(g^{a^{\ell+1}}, h)$ for a randomly chosen $a \in \mathbb{Z}_p^*$. Formally we define:

$$Suc_A^{\ell-CDH}(k) = |Pr[A(h, g, g^a, g^{a^2}, \ldots, g^{a^\ell}) = e(g^{a^{\ell+1}}, h)]$$

Definition 23. *(ℓ-CDH). We say that the ℓ-CDH assumption holds in bilinear groups $\mathbb{G}, \mathbb{G}_1$ if for any ℓ polynomial in k and for any probabilistic polynomial time adversary A, its success in computing $e(g^{a^{\ell+1}}, h)$ after seeing the tuple $(h, g, g^a, g^{a^2}, \ldots, g^{a^\ell})$, i.e. $Suc_A^{\ell-CDH}(k)$, is negligible in k.*

Decisional Diffie-Hellman Assumption

Let $\mathbb{G}$ be a group of prime order p and let g be a random generator for $\mathbb{G}$. In the DDH problem the adversary is given the tuple (g, g^a, g^b) and a value T and it must decide whether $T = T_0 = g^{ab}$ or $T = T_1$ where T_1 was chosen uniformly at random from $\mathbb{G}$ for a randomly chosen $a, b \in \mathbb{Z}_p^*$. Formally we define:

$$Adv_A^{DDH}(k) = |Pr[A(g, g^a, g^b, T_0) = 0] - Pr[A(g, g^a, g^b, T_1) = 0]|$$

Definition 24. *(DDH). We say that the DDH assumption holds in the group $\mathbb{G}$ if for any probabilistic polynomial time adversary A, its advantage in distinguishing Diffie-Hellman tuples, $Adv_A^{BDH}(k)$, is negligible in k.*

See [13] for candidates for DDH groups.

Gap Diffie-Hellman Groups

Let $\mathbb{G}$ be a group of prime order p with a generator g. We consider the two problems specified above, Decisional Diffie-Hellman (Section 11.4) and the Computational Diffie-Hellman (Section 11.4). A group $\mathbb{G}$ is considered a gap Diffie-Hellman group if the *decisional* DH problem is "easy" in that group (polynomial time) and the *computational* DH is "hard" in $\mathbb{G}$ (cannot succeed in computing the target value in polynomial time with more than negligible probability). More formally:

Definition 25. *(Gap Diffie-Hellman groups). We say that $\mathbb{G}$ is a (τ, t, ε)-gap Diffie-Hellman group if the decisional Diffie-Hellman can be computed in time τ and no probabilistic t-time algorithm can successfully compute the computational Diffie-Hellman problem with probability greater than ε, i.e.*

$$Pr_{a,b \xleftarrow{R} \mathbb{Z}_p^*}[A(g, g^a, g^b) = g^{ab}] \leq \varepsilon$$

We say that a group $\mathbb{G}$ is a GDH group if for any polynomial t there exists a negligible function ε and a polynomial τ such that $\mathbb{G}$ is a (τ, t, ε)-gap Diffie-Hellman group.

Candidates for GDH are groups on supersingular elliptic curves or hyperelliptic curves over a finite field, while the bilinear pairings can be derived from the Weil or Tate pairings. For more details see [48].

ℓ-Diffie-Hellman Inversion Assumption

Let $\mathbb{G}$ be a group of prime order p and let g be a random generator for $\mathbb{G}$. In the ℓ-DHI problem the adversary is given the tuple $(g, g^a, g^{a^2}, \ldots, g^{a^\ell})$ and needs to compute $g^{1/a}$ for a randomly chosen $a \in \mathbb{Z}_p^*$. Formally we define:

$$Suc_A^{DHI}(k) = |Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^\ell}) = g^{1/a}]$$

Definition 26. *(ℓ-DHI). We say that the ℓ-DHI assumption holds in the group $\mathbb{G}$ if for any ℓ polynomial in k and for any probabilistic polynomial time adversary A, its success in computing $g^{1/a}$ after seeing the tuple $(g, g^a, g^{a^2}, \ldots, g^{a^\ell})$, i.e. $Suc_A^{DHI}(k)$, is negligible in k.*

ℓ-Bilinear Diffie-Hellman Assumption

Let $\mathbb{G}, \mathbb{G}_1$ be two groups of prime order p equipped with the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ and let g and h be two random generators for $\mathbb{G}$. In the ℓ-BDH problem the adversary is given the tuple $(g, h, g^a, g^{a^2}, \ldots, g^{a^\ell})$

and a value T and it must decide whether $T = T_0 = e(g,h)^{1/a}$ or $T = T_1$ where T_1 was chosen uniformly at random from $\mathbb{G}_1$ for a randomly chosen $a \in \mathbb{Z}_p^*$. Formally we define:

$$Adv_A^{BDH}(k) = |Pr[A(g, h, g^a, g^{a^2}, \ldots, g^{a^\ell}, T_0) = 0] - Pr[A(g, h, g^a, g^{a^2}, \ldots, g^{a^\ell}, T_1) = 0]|$$

Definition 27. *(ℓ-BDH). We say that the decisional ℓ-BDH assumption holds in bilinear groups $\mathbb{G}, \mathbb{G}_1$ if for any ℓ polynomial in k and for any probabilistic polynomial time adversary A, its advantage, $Adv_A^{BDH}(k)$, is negligible in k.*

ℓ-Decisional Bilinear Diffie-Hellman Inversion Assumption

Let $\mathbb{G}, \mathbb{G}_1$ be two groups of prime order p equipped with the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ and let g be a random generator for $\mathbb{G}$. In the ℓ-DBDHI problem the adversary is given the tuple $(g, g^a, g^{a^2}, \ldots, g^{a^\ell})$ and a value T and it must decide whether $T = T_0 = e(g,g)^{1/a}$ or $T = T_1$ where T_1 was chosen uniformly at random from $\mathbb{G}_1$ for a randomly chosen $a \in \mathbb{Z}_p^*$. Formally we define:

$$Adv_A^{DBDHI}(k) = |Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^\ell}, T_0) = 0] - Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^\ell}, T_1) = 0]|$$

Definition 28. *(ℓ-DBDHI). We say that the decisional ℓ-DBDHI assumption holds in bilinear groups $\mathbb{G}, \mathbb{G}_1$ if for any ℓ polynomial in k and for any probabilistic polynomial time adversary A, its advantage, $Adv_A^{DBDHI}(k)$, is negligible in k.*

Decisional ℓ-Weak Bilinear Diffie-Hellman Inversion Assumption

Let $\mathbb{G}, \mathbb{G}_1$ be two groups of prime order p equipped with the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ and let g and h be two random generators for $\mathbb{G}$. In the ℓ-wBDHI problem the adversary is given the tuple $(g, h, g^a, g^{a^2}, \ldots, g^{a^\ell})$ and a value T and it must decide whether $T = T_0 = e(g,h)^{a^{\ell+1}}$ or $T = T_1$ where T_1 was chosen uniformly at random from $\mathbb{G}_1$ for a randomly chosen $a \in \mathbb{Z}_p^*$. Formally we define:

$$Adv_A^{wBDHI}(k) = |Pr[A(g, h, g^a, g^{a^2}, \ldots, g^{a^\ell}, T_0) = 0] - Pr[A(g, h, g^a, g^{a^2}, \ldots, g^{a^\ell}, T_1) = 0|$$

Definition 29. *(ℓ-wBDHI). We say that the decisional ℓ-wBDHI assumption holds in bilinear groups $\mathbb{G}, \mathbb{G}_1$ if for any ℓ polynomial in k and for any probabilistic polynomial time adversary A, its advantage, $Adv_A^{wBDHI}(k)$, is negligible in k.*

ℓ-Decisional Diffie-Hellman Exponent Assumption

Let $\mathbb{G}, \mathbb{G}_1$ be two groups of prime order p equipped with the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ and let g and h be two random generators for $\mathbb{G}$.
In the ℓ-DDHE problem the adversary is given the tuple $(g, h, g^a, g^{a^2}, \ldots, g^{a^{\ell-1}}, g^{a^{\ell+1}}, \ldots, g^{a^{2\ell}})$ and a value T and it must decide whether $T = T_0 = e(g,h)^{a^\ell}$ or $T = T_1$ where T_1 was chosen uniformly at random from $\mathbb{G}_1$ for a randomly chosen $a \in \mathbb{Z}_p^*$. Formally we define:

$$Adv_A^{DDHE}(k) = |Pr[A(g, h, g^a, g^{a^2}, \ldots, g^{a^{\ell-1}}, g^{a^{\ell+1}}, \ldots, g^{a^{2\ell}}, T_0) = 0] - Pr[A(g, h, g^a, g^{a^2}, \ldots, g^{a^{\ell-1}}, g^{a^{\ell+1}}, \ldots, g^{a^{2\ell}}, T_1) = 0]|$$

Definition 30. *(ℓ-DDHE). We say that the decisional ℓ-DDHE assumption holds in bilinear groups $\mathbb{G}, \mathbb{G}_1$ if for any ℓ polynomial in k and for any probabilistic polynomial time adversary A, its advantage, $Adv_A^{DDHE}(k)$, is negligible in k.*

Discussion

We now discuss the relationships between the different assumptions, which assumption implies the other and on which assumptions it seems sensible to rely on. Consider two problems P_1 and P_2 such that given a polynomial time algorithm B that solves P_2, one can design a polynomial time algorithm A that solves P_1 and uses B as a subroutine. In this case we say that $P_1 \preceq P_2$, which means that the problem P_2 is harder than P_1, as one can solve the problem P_1 using an algorithm for solving P_2, but not necessarily the other way around. We also say in that case that the assumption based on the problem P_1 (i.e. the assumption stating that the problem P_1 is hard to solve in polynomial time) is stronger than that of P_2. When $P_1 \preceq P_2$ holds, it also means that if the P_1 assumption holds than so does the P_2 assumption. Note that we are looking for **weak** assumptions to use for our constructions and not strong ones, as weak problems are harder to solve which makes them more likely to hold in practice.

Out of all the assumptions we described, some have reductions to one another, thus we can state which one is stronger than the other. Some assumptions don't have reductions (at least not obvious or known ones) as some assumptions don't relate well to each other. We have the factoring based assumptions, which are: factoring, RSA hardness assumption, strong RSA hardness assumption and the RSA$'$ $s(k)$-hardness assumption. All these factoring assumptions have a reduction to factoring, making it the weakest of all these assumptions. Naturally the strong RSA hardness assumption reduces to the RSA hardness assumption. Unlike the rest of the factoring based assumptions, the RSA$'$ $s(k)$-hardness assumption is unfalsifiable (as far as we know) and is not a cryptographically solid assumption, which we would rather not use.

Consider the unparameterized assumptions over groups: discrete logarithm assumption (DL), computational (CDH) and decisional Diffie-Hellman (DDH) assumptions and Gap Diffie-Hellman groups (GDH). One could show the following reductions over the same group:

1. $DDH \preceq CDH$.
2. $GDH \preceq CDH$.
3. $CDH \preceq DL$.

Thus the hardest problem is DL, which makes the DL assumption the weakest, where the CDH assumption follows as it is weaker than both the DDH and GDH assumptions. The GDH and DDH assumptions are closely related, as the groups where the GDH assumption holds are groups where the DDH assumption doesn't hold and vice versa (as GDH groups are groups where the DDH problem is "easy"). Candidates for GDH groups (i.e. groups where the DDH problem is known to be "easy") are groups on supersingular elliptic curves or hyperelliptic curves over a finite field, while the bilinear pairings for this problem can be derived from the Weil or Tate pairings. For more details see [48]. Boneh [13] suggests in his paper candidates for DDH groups.

The remaining problems, which are the parametrized ones are: ℓ-Computational Diffie-Hellman Assumption (ℓ-CDH), ℓ-Diffie-Hellman inversion (ℓ-DHI), ℓ-Bilinear Diffie-Hellman (ℓ-BDH), ℓ-decisional Bilinear Diffie-Hellman inversion (ℓ-DBDHI), decisional ℓ-weak Bilinear Diffie-Hellman inversion (ℓ-wBDHI) and ℓ-decisional Diffie-Hellman exponent (ℓ-DDHE). For which we can show the following seven reductions in order to help us map their difficulty:

1. $\ell - DBDHI \preceq \ell - BDH$.
2. $\ell - BDH \preceq \ell - DHI$.
3. $\ell - DHI \preceq CDH$.
4. $\ell - DBDHI \leq \ell - wBDHI$.
5. $\ell - wBDHI \leq \ell - CDH$.
6. $\ell - CDH \leq CDH$.
7. $\ell - DDHE \preceq (\ell - 1) - wBDHI$.

From those reductions we can understand a little how they relate to each other. Using these reductions we sort the assumptions from strong to weak, we have two chains of reductions: $\ell - DBDHI, \ell - BDH, \ell - DHI, CDH$ and $\ell - DBDHI, \ell - wBDHI, \ell - CDH, CDH$, while we know that the $\ell - DDHE$ assumption

is weaker than the $(\ell - 1) - wBDHI$ assumption. Note that clearly we would like to use an assumption where the parameter ℓ is as small as possible, as the smaller the parameter the greater the chance that the assumption holds in practice. Boneh et al. prove that the ℓ-DBDHI assumption holds in the generic group model [73]. 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. Using the reductions above, this result extends to the rest of the parametrized assumptions, besides the $\ell - DDHE$ assumption, thus making them more likely to hold in practice. This concludes what we know about the assumptions used for constructions in this paper.

11.5 Hoeffding Inequality

We use the following inequality proved by Hoeffding [42]:

$$Pr[E[S_n] - S_n \geq t] \leq exp(-\frac{2t^2}{\sum_{i=1}^{n}(a_i - b_i)^2})$$

S_n is a sum of independent identically distributed indicators X_i, where $X_i = 1$ if we learn secret i and 0 otherwise. a_i and b_i are lower and upper bounds on the values of the random variables, i.e. $a_i = 0$ and $b_i = 1$, because they are indicators. As we mentioned before the probability for a *secondary* to learn secret i is at most $\frac{1}{2}$, thus $E[X_i] \leq \frac{1}{2}$. We choose $t = \frac{n}{6}$ to get:

$$Pr[\frac{n}{2} - S_n \geq \frac{n}{6}] = Pr[S_n \leq \frac{n}{3}] \leq exp(-\frac{2(\frac{n}{6})^2}{\sum_{i=1}^{n}(a_i - b_i)^2}) = e(-\frac{n}{18}) = neg(n)$$