Distributed Pseudo-Random Functions and KDCs

Moni Naor[⋆] Benny Pinkas[⋆⋆] Omer Reingold[⋆⋆⋆]

Dept. of Computer Science and Applied Math
Weizmann Institute of Science
Rehovot, Israel

Abstract. This work describes schemes for distributing between n servers the evaluation of a function f which is an approximation to a random function, such that only authorized subsets of servers are able to compute the function. A user who wants to compute $f(x)$ should send x to the members of an authorized subset and receive information which enables him to compute $f(x)$. We require that such a scheme is consistent, i.e. that given an input x all authorized subsets compute the same value $f(x)$.
The solutions we present enable the operation of many servers, preventing bottlenecks or single points of failure. There are also no single entities which can compromise the security of the entire network. The solutions can be used to distribute the operation of a Key Distribution Center (KDC). They are far better than the known partitioning to domains or replication solutions to this problem, and are especially suited to handle users of multicast groups.

1 Introduction

A single server that is responsible for a critical operation is a performance bottleneck and a single point of failure. A common approach for solving this problem is the use of several replicated servers. However this type of solutions degrades the security of the system if the servers should store secrets (e.g. keys) which are required for cryptographic operations. A solution to both the availability and the security problems is to design a system whose security is not affected if a limited number of servers are broken into (see Section 1.2 for a discussion of the availability and security issues for KDCs).

The problem of distributing the evaluation of trapdoor functions for public key cryptography was extensively investigated (see e.g. [18, 17, 22, 42]). However, the problem of distributing the functions needed for private key cryptography,

⋆ Research supported by an infrastructure research grant of the Israeli Ministry of Science. Email: naor@wisdom.weizmann.ac.il

⋆⋆ Research supported by an Eshkol Fellowship of the Israeli Ministry of Science. Email: bennyp@wisdom.weizmann.ac.il

⋆⋆⋆ Research supported by a Clore Scholars award and an Eshkol Fellowship of the Israeli Ministry of Science. Email: reingold@wisdom.weizmann.ac.il

in particular the distribution of the evaluation of pseudo-random functions, was neglected (an exception is the work of [31]). Threshold evaluation of random-like functions is required for seemingly unrelated applications, for example for secure and efficient metering of web usage [32], for threshold evaluation of the Cramer-Shoup cryptosystem [13], and for the applications we discuss in this paper (in particular, distributed KDCs and long-term repository for encrypted data). These applications require that the protocol for the collective function evaluation does not invovle communication between the parties which evaluate the function. This requirement is not satisfied by most threshold constructions for public key cryptography.

This work describes schemes for distributing between n servers the evaluation of a function f which is an approximation to a random function, such that only authorized subsets of servers are able to compute the function. A user who wants to compute $f(x)$ should send x to the members of an authorized subset and receive information which enables him to compute $f(x)$. We require that such a scheme is consistent, i.e. that given an input x all authorized subsets compute the same value $f(x)$.

Distributed and consistent evaluation of pseudo-random functions is useful for many applications. The consistency property is especially useful for the following three types of applications:

(i) A distributed KDC system (DKDC), in particular for multicast communication. We describe this application in detail in Section 1.2.

(ii) *Long-tem encryption of information*, where a user might want to encrypt personal information and keep the decryption keys safely distributed between many servers (see Section 1.3).

(iii) A realization of a *Random Oracle* or of a beacon [41] that generates randomness which should be shared by remote parties and used in a cryptographic protocol.

We introduce the notion of a *Distributed Pseudo-Random Function (DPRF)*. We describe several constructions of approximations to random functions which are useful for many of the applications of a DPRF. A *threshold DPRF* (depicted in Figure 1) is a system of n servers such that *any* k of them can compute the function f, but breaking into any $k-1$ servers does not give any information about f (for instance think of a system with $n = 20$ servers and a threshold of $k = 3$). The servers could be distributed across the network, and a party can contact any k of them in order to compute f. If several parties need to compute f for the same input they are not required to contact the same k servers but rather each party can contact a *different* set of k servers (e.g. those to which it has the best communication channels). Furthermore, to reduce the latency of the computation a party can contact the k servers *in parallel.* We also support DPRFs based on general monotone access structures [7, 28, 3] rather than on threshold ones. There are several scenarios where general access structures might be preferable to threshold access structures (e.g. to allow efficient implementations of quorum systems [38] which enable fast revocation).

Our constructions can be further amended to be robust against servers which

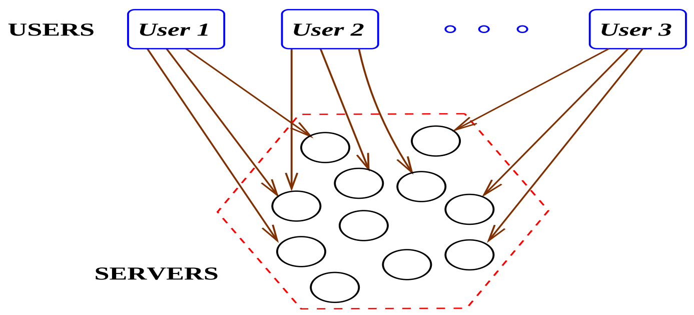

Fig. 1. A Distributed Pseudo-Random Function System.

send incorrect data to users who approach them, (the robustness is based either on error-correcting mechanisms or on proof techniques). The constrcutions can also be further improved to ensure *proactive security* (see [11] and references therein for a general discussion of proactive security), which provides automatic recovery from break-ins: The servers perform some periodic refreshment of their secrets (e.g. once a day), and as a result only an adversary which breaks into k servers in the *same period* can break the security of the system.

1.1 Our Solutions for a Threshold Access Structure

It is unknown how to perform a threshold evaluation of a pseudo-random function without requiring heavy communication between the servers for each given input. Lacking a general construction we describe three different approximations of DPRFs with a threshold access structure.

The first construction generates f as an ℓ-wise independent function. It provides information theoretic security as long as an adversary does not obtain ℓ different values. The scheme is very efficient and requires only multiplications in a *small* finite field (which essentially should only be large enough so that a random element in it can be used as a key for a private-key encryption scheme). The parameter ℓ can therefore be set to be rather large (even several millions).

The second construction is based on a computational assumption: the decisional Diffie-Hellman assumption (see [9]). However the resulting function is only weakly pseudo-random, i.e. it is pseudo-random as long as the inputs on which it is evaluated are pseudo-random. The construction requires a user to compute $O(k)$ exponentiations in order to compute the function's output, and a server should compute only a single exponentiation in order to serve a user. The first two constructions can be easily amended to provide proactive security.

The third construction is based on a monotone CNF formula realizing the threshold k-out-of-n function. This construction computes a full-fledged pseudo-random function and its security depends only on the existence of pseudo-random functions. It can also be adapted to any access structure. Its drawback is that it is only efficient for moderate values of n and small values of k, and we do not know how to enhance it to obtain proactive security.

The constructions and their properties are summarized in Table 1.

	Efficiency	Pseudo randomness	Number of evaluations	Proactive security	Robust.	General access
ℓ-wise ind.	efficient poly	strong	limited	yes	yes	yes
DDH	expensive poly	weak	unlimited	yes	yes	yes
CNF	exponential	strong	unlimited	no	yes	yes

Table 1. A comparison of the threshold schemes.

DPRFs for general access structures: We present constructions of DPRFs based on any monotone access structure. For example, an access structure based on a quorum system allows for fast user revocation by accessing the servers which are members of a single quorum. Our constructions are based either on monotone symmetric branching programs (contact schemes), or on monotone span programs.

1.2 Application to Key Distribution – DKDCs

A Key Distribution Center (KDC): A popular approach for generating common keys between two parties without using public key cryptography is by using a three-party trust model which includes a trusted key distribution center (KDC). In networks which use a KDC there is a dedicated key between the KDC and each of the members of the network. Denote by k_u the key between the KDC and party u. This is the only key that u has to store. Very informally, when two parties (e.g. u and v) wish to communicate, one of them approaches the KDC which then provides a random key, $k_{u,v}$, and sends it to each of the two parties, encrypted with their respective secret keys (i.e. $E_{k_u}(k_{u,v})$ (the encryption of $k_{u,v}$ with the key k_u) is sent to party u, and $E_{k_v}(k_{u,v})$ is sent to v). The parties can now communicate using the key $k_{u,v}$. This approach was initiated by Needham and Schroeder in 1978 [40] and is widely implemented, most notably in the Kerberos system (see e.g. [27]). Bellare and Rogaway [4] give a complexity-theoretic treatment of this model, and present a provably secure protocol for session key distribution based on the existence of pseudo-random functions.

The approach of using a KDC is appealing since each party should only store a single key and when a new party is introduced there is no need to send keys to other parties. However there are various problems in using KDCs, which are due to the fact that a KDC is a single point of failure:

– **Security:** The KDC knows all the keys that are used in the system, and if it is broken into the security of the entire network is compromised.
– **Availability:** (i) The KDC is a performance bottleneck, every party has to communicate with it each time it wishes to retrieve a key. (ii) When the KDC is down or unreachable no party can obtain new keys for starting

conversations on the network. (iii) The availability problem is amplified when trying to use a KDC to generate keys for multicast communication (i.e. to be shared by more than two parties), since all the relevant parties have to contact a single KDC.

In order to address these problems the common practice is to use multiple KDCs. However, the known solutions are far from being perfect: **(i)** The security problem is addressed by dividing the network into different *domains* and dedicating a different KDC to each domain. When a KDC is broken into only the domain to which it belongs is compromised. However, the management of inter-domain connections is complicated and a KDC still holds all the secrets of its domain. **(ii)** The availability problem is reduced by replicating the KDC and installing several servers each containing all the information that was previously stored in the KDC. This improves the availability but decreases security: there are multiple sensitive locations and breaking into *any* of these replicated KDCs compromises the security of the network. There is also an additional problem of reliably synchronizing the information that is stored in the different copies.

Multicast communication: The availability problem is relevant to unicast communication between two parties but is even more severe for multicast communication. Multicast communication is sent to a (potentially large) number of parties. Typical applications are the transmission of streams of data (such as video streams) to large groups of recipients, or an interactive multiparty conference. The large (exponential) number of groups in which a party might participate prevents it from storing a key for each potential group. On the other hand, the large number of parties which might require the service of the KDC worsen the availability problem. For example, imagine a source which transmits many video channels over the Internet, with hundreds of thousands of receivers all over the world. A *single* KDC cannot handle requests from all these receivers. Alternatively, consider a multinational company which uses a single KDC for providing keys for virtual meetings of its employees. If some offices are disconnected from the KDC then users in these offices cannot even obtain keys for virtual meetings between themselves.

A Distributed KDC – DKDC: A DPRF can be used to construct a Distributed KDC (DKDC). A DKDC consists of n servers and a user should k of them in order to obtain a key. The servers are responsible for a consistent mapping between key names and key values[4]. Each KDC server should operate as a server in the distributed evaluation of the pseudo-random function f. The key for a certain subset S of users is defined as $k_S = f(S)$. This approach is especially useful for generating keys for multicast groups with many members. Each member might approach a different authorized subset of the KDCs and it is guaranteed that every user obtains the same key. It is also useful to use

[4] Of course, consistency does not prevent groups of users from using different keys at different times (session keys), if this is desired.

this construction to generate keys for unicast communication if each of the two parties prefers to access a different subset of KDCs.

Key granting policy: When a user requests a key from a KDC the KDC should decide whether the user is entitled to receive this key. The question of how this decision is made is independent of this work.

One appealing approach is when a group name is derived from the identities of its members and then servers can easily verify whether a user that asks for a key is part of the group of users that use the key. This method is good for "mid-sized" groups. For larger groups the group name can be generated by a method based on hash trees, and then a user can efficiently prove to a server that it is part of a group.

Another appraoch introduces an interesting billing mechanism for multicast transmissions with k-out-of-n DKDCs: the user is required to pay each server $1/k$ of the payment needed for accessing the transmission, and to receive in return the information the server can contribute towards the reconstruction of the decryption key.

1.3 Long-term Encryption of Information

Suppose one wishes to store encrypted information so that it remains safe for many years. A problem that immediately arises is where to store the keys used for the encryption so that they would not be leaked or lost. Note that the question of storing keys safely arises in many other scenarios, e.g. [8]. One possibility is to use a DPRF as a long term key repository. We add to the system a collection of n servers that act as the servers of the DPRF. These servers are trusted in the sense that no more than k of them become faulty[5]. We should also have some way to specify the policy determining who is allowed to decrypt the file, as the system is likely to be used by many users. We assume that the DPRF has ways to check whether a user is allowed to obtain information with a given policy (this is orthogonal to the issue at hand).

In order for a user to encrypt a file X and decryption policy specified by who, it does the following

- Choose an encryption key r for a conventional encryption scheme G and encrypt the file with key r. Let $Y = G_r(X)$.
- Compute $h = H(Y)$ where H is a collision intractable hash function.
- Apply to the DPRF to obtain $s = f(h \circ \text{who})$
- Put Y in the long term storage together with who and $s \oplus r$.

To decrypt an encrypted file Y with policy who and encrypted key s':

- Compute $h = H(Y)$.
- Apply the DPRF to obtain $s = f(h \circ \text{who})$
- Decrypt Y with key $s \oplus s'$ to G.

[5] In this case the desirability of proactive security is evident since the assumption is that no more than k are broken into at any given period.

Note that we do not require the servers of the DPRF to store anything in addition to their keys. All information related to the file can be stored at the same place. Also note that in order combat changes to the stored information one should use parts of s as an authentication key to Y and r.

1.4 Related Work

DPRF systems perform multi-party computations. The generic solutions of [25, 6, 14] for multiparty computations are inefficient for this application (even when applied to the relatively simple pseudo-random functions of [36], see discussion there). In particular, they require communication between the servers which are involved in the evaluation of the function. Their security is also only guaranteed if less than one third (or one half in the passive model) of the servers are corrupted.

There has been a lot of work on designing and implementing KDCs. A good overview of this work can be found in [27] and a formal treatment of the problem is given in [4]. Most of this work was for a trusted party which generates a key "on-the-fly", i.e. where consistency of the key is not required. While this model may be more relevant to unicast it is less applicable when more than two parties are involved.

Naor and Wool [39] considered a different scenario for protecting databases, and when adapted to our scenario their solution is one where the servers are trusted never to reveal their secret keys, but some of them might not have received updates regarding the permissions of users (which is a weaker assumption than regarded in this paper).

Our first two constructions are similar in nature to the constructions of Naor and Pinkas for metering schemes [32]. The problem they considered was to enable a server to *prove* that it served a certain number of clients (a representing application might be to meter the popularity of web sites in order to decide on advertisement fees). In general, not every solution for the metering problem is relevant to the construction of a DPRF (for example, the output of the metering computation should be *unpredictable* whereas the output of a DPRF should be *pseudo-random*). The metering constructions achieve better *robustness* against transmission of corrupt *proof components* than the robustness of our DPRF schemes against corrupt *key components*. On the other hand the metering constructions do not provide proactive security (due to the lack of communication channels between clients in that model) whereas we present very efficient proactive enhancements to the DPRF schemes.

Micali and Sidney [31] showed how to perform a shared evaluation of a pseudo-random function with a non-tight threshold. They provided a lower bound and a non-optimal probabilistic construction which is relevant only for small values of k and n. We describe an deterministic construction for the sharp threshold case which matches their lower bound.

Gong [26] considered a problem related to the DKDC application: a pair of users A and B each have private channels to n servers, and would like to use them to send a secret and authentciated message from A to B (e.g. a key which they will later use). Some of the servers might be corrupt and might change the messages they are asked to deliver (this problem is similar to that considered by Dolev et al [19] since each server is essentially a faulty communication link). Gong's scheme requires A to send through each server a message of length $O(n)$

2 Definitions

2.1 The Model

The following model is used throughout this work.
Setting: We consider a network of many users (clients), which also contains n servers $S_1 \ldots, S_n$. Each user u has a private connection with each of at least k servers (in all but the proactive solutions these channels can be realized using symmetric encryption. A future work [34] describes how to efficiently maintain these channels in the proactive model).
Initialization: At the initialization stage each server S_i receives some secret personal key α_i which it would use in its subsequent operation. It is possible that the values $\{\alpha_i\}_{i=1}^n$ were generated by a central authority from a system key α. If this is the case then α is erased at the end of the initialization stage. Preferably, the servers perform a short joint computation which generates the values $\{\alpha_i\}_{i=1}^n$, such that no coalition C of $k-1$ servers can use its values to learn anything about α_u if $u \notin C$. This prevents even a temporary concentration of the system's secrets at a single location.
Regular operation: A party u that wants to compute $f(x)$, operates as follows:

- It contacts k servers, $S_{i_1}, \ldots, S_{i_k}$, and sends to each of them a message $\langle u, x\rangle$.
- Each server S_i verifies that u is entitled to compute $f(x)$. If so, it computes a function $F(\alpha_i, x)$, and sends the result to u through their private channel.
- u computes $f(x)$ from the answers it received using a function G, namely it computes $f(x) = G(h, F(\alpha_{i_1}, x), \ldots, F(\alpha_{i_k}, x))$.

2.2 Requirements

There are two approaches to approximating random functions: *pseudo-randomness* and *ℓ-wise independence*. We present approximations to DPRFs which follow both these directions.

Loosely speaking, pseudo-random distributions cannot be efficiently distinguished from uniform distributions. However, pseudo-random distributions have substantially smaller entropy than uniform distributions and are efficiently sampleable. *Pseudo-random function ensembles*, which were introduced in [24], are distributions of functions. These distributions are indistinguishable from the uniform distribution under all (polynomially-bounded) black-box attacks (i.e. the distinguisher can only access the function by adaptively specifying inputs and getting the value of the function on these inputs). Goldreich, Goldwasser, and Micali provided a construction of such functions based on the existence of pseudo-random generators. See [23, 29] for further discussions and exact definitions of pseudo-random functions.

We also use *ℓ-wise independent functions*. Their difference from a pseudo-random function is that more than ℓ values of an ℓ-wise independent function are not "random looking" (however, a set of at most ℓ values is completely random rather than pseudo-random).

In a DPRF the ability to evaluate the function is distributed among the servers. The process that is performed by the servers can be defined as *k-out-of-n threshold function evaluation*.

Definition 1 k-out-of-n threshold evaluation of a pseudo-random func. Let $\mathcal{F}_m = \{f_\alpha\}$ be a family of pseudo-random functions with security parameter m, keyed by α. A k-out-of-n computation of $\mathcal{F}_m$ is a triple of polynomial time functions $\langle S, F, G \rangle$ (the key sharing, share computation and construction functions), such that

- For every $f_\alpha \in \mathcal{F}_m$, $S(\alpha) = \langle \alpha_1, \ldots, \alpha_n \rangle$, such that
- For every $1 \leq i_1 < \cdots < i_k \leq n$, $G(\langle i_1, F(\alpha_{i_1}, x)\rangle, \ldots, \langle i_k, F(\alpha_{i_k}, x)\rangle) = f_\alpha(x)$. And,
- For every $1 \leq i_1 < \cdots < i_{k-1} \leq n$, given $\{\alpha_{i_j}\}_{j=1}^{k-1}$, and given a set Y of polynomially many values (where the inputs in Y were chosen adaptively, possibly depending on $\{\alpha_{i_j}\}_{j=1}^{k-1}$), and the values $\langle f_\alpha(y), \{F(\alpha_i, y)\}_{i=1}^{n}\rangle$ for every $y \in Y$, the restriction of the function f_α to inputs which are not in Y is pseudo-random.

The definition of k-out-of-n threshold evaluation of an ℓ-wise independent function is similar, except that $\mathcal{F}_m$ is a family of ℓ-wise independent functions, and it is required that given the computation process of any $\ell - 1$ function values, any remaining value is uniformly distributed.

The most important requirement of k-out-of-n threshold function evaluation is that the output of f be consistent. The protocol might be considered as a special case of multi-party computations [25, 6, 14]. However although it might not be obvious from first reading, our definition includes several efficiency restrictions which do not exist in the definition of multi-party computations and which are actually not satisfied by the constructions of [25, 6, 14] (their constructions are also for a joint computation by n parties, and are secure only against coalitions of less than $n/2$ or $n/3$ parties. Our requirement is for a joint computation by k parties and security against $k-1$ servers, where k might be any number up to n). The efficiency requirements, which we explicitly state below, are needed to minimize the communication overhead which is often the most important factor of the system's overhead. The efficiency requirements are:
Communication pattern: In the process of computing $f(x)$ there is no communication between the servers. The only communication is between the servers and the party that computes $f(x)$.
Single round: There is only a single round of communication between the servers and the user. The user can send queries to the servers in parallel, i.e. there is no need to wait for the answer from one server before sending a query to another server.

Obliviousness: The query to one server does not depend on the identities of the other servers which the user queries. This requirement is important if the user might find (while in the middle of the process of querying the servers) that some of the servers to which it applied are malfunctioning.

Additional requirements can be considered as security optimizations to the original definition. They are not obligatory, but improve the quality of a DPRF construction:

Robustness: If a server is controlled by an adversary it might send to the user corrupt information which prevents the user from computing the correct value. It is preferable if the user can identify when such an event happens.

Proactive security (or, Resilience to prolonged attacks): Proactive security enables a system to maintain its overall security even if its components are repeatedly broken into. Systems with proactive security typically use a security parameter k and are secure as long as less than k system components are broken into *in the same time period* (see [11] for a discussion of proactive security).

3 The Threshold Constructions

3.1 ℓ-wise Independence based on Bivariate Polynomials

The first construction is based on a generalization of the secret sharing scheme of Shamir [44] to bivariate polynomials. It is a threshold construction of an ℓ-wise independent function. The scheme can be used to generate more than ℓ values as long as it is guaranteed that no adversary will get hold of ℓ values. It is not necessarily decided in advance which values will be generated by the scheme.

Setting: The family $\mathcal{F}$ is the collection of all bivariate polynomials $P(x, y)$ over a finite field $\mathcal{H}$, in which the degree of x is $k-1$ and the degree of y is $\ell-1$. The key α defines an element $f_\alpha \in \mathcal{F}$ (α consists of the $k\ell$ coefficients of the polynomial). The output of the function is an element in the field $\mathcal{H}$. All the arithmetic operations performed by the scheme are over $\mathcal{H}$.

Initialization: (we describe here an initialization by a central authority, later we also describe how the servers can perform a distributed initialization). The initializer of the system chooses a random key α which defines a random polynomial $P(x, y)$ from $\mathcal{F}$. Each server S_i receives the key $\alpha_i = Q_i(y) = P(i, \cdot)$, which is an $\ell - 1$ degree polynomial in y.

Operation: The value $f(h)$ is defined as $f(h) = P(0, h)$. Consider a user that wishes to compute this value. Say the user approaches server S_i, then it should send him the information $\beta_{i,h} = F(\alpha_i, h) = Q_i(h) = P(i, h)$. After receiving information from k servers $S_{i_1}, \ldots, S_{i_k}$ the user can perform a polynomial interpolation through the points $\{\langle i_j, \beta_{i_j,h}\rangle\}_{j=1}^{k}$ and compute the free coefficient of the polynomial $Q_h(x) = P(\cdot, h)$, namely the value $f(h) = P(0, h)$.

The following points can be easily verified: **(i)** The scheme implements the definition of k-out-of-n evaluation of an ℓ-wise independent function. **(ii)** In a DKDC application the size of an element in the field $\mathcal{H}$ should be the length of the required key and can therefore be rather small (e.g. 128 bits). The scheme can be therefore used to produce a large number of keys (e.g. $\ell = 10^6$).

Several modifications can enhance the above scheme: **(i)** Proactive security can be easily obtained, see Section 5. **(ii)** In order to reduce the complexity of the polynomial interpolation it is possible to use several polynomials of smaller degree and map keys to polynomials at random. **(iii)** It is possible to perform a distributed initialization of the polynomial P, and then the system's secrets are never held by a single party. The initialization is performed by several servers which each define a bivariate polynomial, and the polynomial used by the system is the sum of these polynomials. Only a coalition of all these servers knows shares of other servers. The initialization uses a new verification protocol we discuss in Section 5.

Robustness: A simple and straightforward procedure to verify that a user is receiving correct information from servers, it to require the user to get shares from $k' > k$ servers and use the error-correction properties of Reed-Solomon codes to construct the correct share (see e.g. [30]).

3.2 Distributed Weak PRFs Based on the DDH Assumption

In this section we describe a different kind of approximation for a DPRF: we show a way to distribute a *weak pseudo-random function* [35, 37]. A function f is a weak PRF if it is indistinguishable from a truly random function to a (polynomial-time) observer who gets to see the value of the function on any polynomial number of *uniformly chosen inputs* (instead of any inputs of its choice). The definition of k-out-of-n threshold evaluation of a weak pseudo-random function f is similar to Definition 1. The only difference is that we require that given the computation process of f on any polynomial number of *uniformly chosen inputs*, the value of f on any *additional uniformly chosen input* is indistinguishable from random (this implies that f remains a *weak* pseudo-random function).

The main advantage of a distributed weak PRF compared with distributed ℓ-wise independent function is that the former is secure even when the adversary gets hold of *any polynomial number of values*. However, constructing a distributed weak PRF requires some computational intractability assumption (in particular, the existence of one-way functions). The specific construction described here relies on the decisional version of the Diffie-Hellman assumption (which we denote as the DDH assumption). This construction is rather attractive given its simplicity.

The applicability of weak pseudo-random function: Any distributed weak pseudo-random function f can be transformed to a DPRF by defining $f'(x) = f(RO(x))$, where RO is a random oracle (i.e., a random function that is publicly accessible to all parties as a black-box; see [4]). Therefore, if one postulates the existence of random oracles then the construction we present below can be used for all the applications of DPRFs. However this construction may be applicable even without the use of random oracles. Consider for example the application of DKDCs for multicast communication. Here there may be several scenarios where a distributed weak pseudo-random function is sufficient. One such scenario is when there exists a *public* mapping H that assigns random names to groups of users. The key of a group can be the value of the distributed function applied to the group's name. It is conceivable that group names are chosen by some trusted party (or by a distributed protocol between several parties), and kept in some (possibly duplicated) publicly available server. In fact, using the specific functions described below is secure as long as some member of the group chooses the group name as g^r and proves that it knows r.

In the scheme we describe below, the user who computes the function f should perform k exponentiations. This overhead is larger than that of a Diffie-Hellman key exchange. However, the overhead is justified even for the DKDC application, since the Diffie-Hellman key exchange protocol cannot be used to solve the availability and the security requirements that underline our solution of a consistent distribution of a KDC (and are especially important for multicast communication).

Related distributed solutions were previously suggested for discrete-log based signatures (e.g. [22]). The novelty in our work is the fact that we prove the *pseudo-randomness* of the evaluated function.

Setting and Assumptions: The scheme is defined for two large primes P and Q such that Q divides $P-1$, and an element g of order Q in Z_P^*. The values P, Q and g are public and may either be sampled during the initialization or fixed beforehand. We assume that for these values, the decisional version of the Diffie-Hellman assumption (DDH-Assumption) holds. I.e., that given a uniformly distributed pair $\langle g^a, g^b \rangle$, it is infeasible to distinguish between $g^{a \cdot b}$ and a uniformly distributed value g^c with non-negligible advantage. For a survey on the application of the DDH-Assumption and a study of its security see [9].

The functions and their initialization: The family $\mathcal{F}$ is keyed by a uniformly distributed value $\alpha \in Z_Q^*$. For simplicity, we define the function f_α over $\langle g \rangle$ (where $\langle g \rangle$ denotes the subgroup of Z_P^* generated by g)[6]. The function f_α is defined by $\forall x \in \langle g \rangle,\ f_\alpha(x) \overset{\text{def}}{=} x^\alpha \bmod P$.

The value α is shared between the servers using the secret sharing scheme of Shamir [44]: The initializer of the system chooses a random polynomial $P(\cdot)$ over Z_Q^* of degree $k-1$ such that $P(0) = \alpha$. Each server S_i receives the key $\alpha_i = P(i)$. To facilitate robustness, the initializer also makes the values g^α and $\{g^{\alpha_i}\}_{i=1}^n$ public. It is also possible to let the servers perform a distributed initialization of f.

Operation: Consider a user that wishes to compute $f_\alpha(h)$ and approaches a set of k servers $\{S_i\}_{i \in J}$. Each such server S_i sends to the user the information $\beta_{i,h} = F(\alpha_i, h) = f_{\alpha_i}(h) = h^{\alpha_i}$. After receiving information from the k servers the user can perform a polynomial interpolation through the points $\{\alpha_i\}_{i \in J}$ in

[6] In fact, one can define f'_α over Z_P^* by setting $f'_\alpha(x) = f_\alpha(x')$ where $x' = x^{(P-1)/Q} \bmod P$. If f_α is a weak PRF then so is f'_α since: (1) If x is uniform in Z_P^* then x' is uniform in $\langle g \rangle$. (2) For any $x' \in \langle g \rangle$ one can efficiently compute a uniformly chosen $((P-1)/Q)$-th root of x'. Computing such roots is possible by a generalization of Tonelli's algorithm presented by Adleman, Manders and Miller (see [2] for a survey on this subject).

the exponent of h. I.e he can compute

$$f_\alpha(h) = h^\alpha = h^{\sum_{i\in J}\lambda_{i,J}\cdot\alpha_i} = \prod_{i\in J} h^{\lambda_{i,J}\cdot\alpha_i} = \prod_{i\in J} f_{\alpha_i}(h)^{\lambda_{i,J}}$$

where all exponentiations are in Z_P^* and the values $\{\lambda_{i,J}\}$ are the appropriate Lagrange coefficients.

It is easy to verify that querying any k servers for the value $f_{\alpha_i}(h)$ results in the same final value $f_\alpha(h)$. Memory requirements from each server are minimal (i.e. storing a single value in Z_Q^*). In order to serve a user each server should perform a single modular exponentiation in Z_P^*. A user is required to perform k modular exponentiation in Z_P^*.
The security of the scheme is proved by the following theorem.

Theorem 2. *If the DDH-Assumption holds then the above scheme is a k-out-of-n threshold evaluation of a weak pseudo-random function.*

Proof Sketch: For clarity, we ignore at first the issue of corrupted servers and just prove that if the DDH-Assumption holds then $\mathcal{F} = \{f_\alpha\}$ is a family of weak pseudo-random functions. Let D be an efficient algorithm that gets the value of f_α on $q-1$ uniformly chosen inputs $x_1, \ldots x_{q-1}$ and distinguishes $f_\alpha(x_q)$ from random with advantage ϵ (where x_q is also uniformly distributed). We construct an algorithm A that breaks the DDH-Assumption:

On input $\langle g^\alpha, g^\beta, z\rangle$, the algorithm A first samples random values $\{r_i\}_{i=0}^{q-1}$ (in $\{1, \ldots Q\}$). Then A invokes D and returns its output on the input $\{\langle q_i, f_\alpha(q_i)\rangle\}_{i=0}^{q-1}$ and the additional pair of values $\langle x_q = g^\beta, z\rangle$. Where for each i, $q_i = g^{r_i}$ (and therefore $f_\alpha(q_i) = g^{\alpha\cdot r_i}$ can be evaluated by A). It is easy to verify that the advantage A has in distinguishing between the case that z is uniform in $\langle g\rangle$ and the case the $z = g^{\alpha\cdot\beta}$ is at least ϵ.

We now need to show that no coalition of $k-1$ corrupt servers $S_{i_1}, \ldots, S_{i_{k-1}}$ can break the threshold scheme. The reason this holds is that such $k-1$ servers can be simulated by the algorithm D described above. To do so, D samples the secret values of the $k-1$ servers (i.e., $\alpha_{i_1}, \ldots, \alpha_{i_{k-1}}$) by itself. Let P be the degree $k-1$ polynomial that interpolates these values and α. Define $\alpha_j = P(j)$. D can evaluate every g^{α_j} using interpolation in the exponent of g and can therefore evaluate all the values $f_{\alpha_j}(q_i)$. $\square$

Robustness: Since the values $\{g^{\alpha_i}\}_{i=1}^n$ are public each server can prove the correctness of any answer $f_{\alpha_i}(x) = x^{\alpha_i}$. This can either be done by a zero-knowledge variant of Schnorr's proof for the value of the Diffie-Hellman function [43, 15] or by the non-interactive version that uses random-oracles.

It is possible to perform a distributed initialization of the scheme, secure against corrupt servers (even if their only goal is to disrupt the operation of the system rather than to learn keys), and to achieve to achieve proactive security for the scheme.

3.3 DPRFs based on Any Pseudo-Random Function

The following scheme can use any family of pseudo-random functions, but since its overhead for the k-out-of-n access structure is $O(n^{k-1})$ it is useful only if the total number of servers n is moderate and the threshold k is small.

Setting: Define $d = \binom{n}{k-1}$, and define the d subsets $\{G_j\}_{j=1}^{d}$ as all the subsets of $n-k+1$ of the n servers.

Let $\mathcal{F}_m$ be a collection of pseudo-random functions with security parameter m. The key α is a d-tuple $\langle a_1, \ldots, a_d \rangle$ of elements from $\{1, \ldots, |\mathcal{F}_m|\}$, and defines a d-tuple $\langle f_{a_1}, \ldots, f_{a_d} \rangle$ of elements from $\mathcal{F}_m$. The function f_α is defined as $f_\alpha(x) = \oplus_{j=1}^{d} f_{a_j}(x)$.

Initialization: A random key α is chosen. We would like that for every $1 \leq j \leq d$, all the servers in subset G_j would receive the key to the function f_{a_j}. Therefore for every server S_i, $\alpha_i = \{a_j | i \in S_j\}$. Note that the union of the keys of any k servers covers α and is therefore sufficient to compute f_α.

Operation: The DPRF system would provide the value $f_\alpha(h) = \oplus_{j=1}^{d} f_{a_j}(h)$. When a user approaches a server S_i, and the server approves of the user computing $f(h)$, it should send to the user the information $\{f_{a_j}(h) | a_j \in \alpha_i\}$. I.e., the server should provide to the user the output of all its functions on the input h. After approaching k servers, the user has enough information to compute $f_\alpha(h)$.

For any coalition of $k-1$ serves there is a subset G_j which does not contain any member of the coalition and thus the coalition members cannot compute f_{a_j}. Therefore it is straightforward to prove that the construction is a k-out-of-n evaluation of a pseudo-random function. The number of functions which each server should be able to compute is $\binom{n-1}{k-1}$, and the total number of functions is $d = \binom{n}{k-1}$. Therefore the scheme cannot be used for systems with a large threshold. However, for a moderate n and a small k the overhead is reasonable (e.g. for $n = 50$ and $k = 4$, $d = 19,600$ and a server should compute $4,606$ functions).

Note that the user receives the value of functions f_{a_j} from more than a single server. Therefore if the user sends to servers the identities of the other servers which it approaches, the communication overhead is reduced if a a simple mapping is used to ensure that the output of each function is sent once. Alternatively the data redundancy can be used to provide robustness against corrupt servers that send incorrect data to users.

Generalization: The scheme can be generalized to any access structure. The construction we used corresponds to a monotone CNF formula which contains all clauses of $n-(k-1)$ out of n elements. A similar formula can be used to realize any access structure. The total number of pseudo-random functions used is the number of clauses in the monotone CNF formula.

Comparison to previous work: Micali and Sidney [31] considered more general access structures: they defined an (n, t, u)-*resilient collection* (with $t < u < n$) which enables any subset of u (out of n) parties to perform the computation, while no subset of t parties has this ability. We are interested in a sharp threshold, which provides the best security, and therefore require that $k = u = t + 1$.

Micali and Sidney proved a lower bound of $\frac{n!(u-t-1)!}{(n-t)!(u-1)!}$ for the number of functions in an (n,t,u)-resilient collection, and used the probabilistic method to show the existence of a construction which is $\ln\binom{n}{t}$ times larger than the lower bound. Our deterministic construction (for the sharp threshold case) matches their lower bound, and is therefore optimal.

4 DPRFs with General Access Structures

4.1 Using Monotone Symmetric Branching Programs

We present here generalizations of the threshold schemes to access structures based on *monotone symmetric branching programs*. In Section 4.2 we describe constructions for access structures based on *monotone span programs*. This is a further generalization in that any linear secret sharing scheme can be simulated by a monotone span program of the same size (the converse is also true, i.e. any monotone span program can be simulated by a linear secret sharing scheme of the same size, see [3]). However, the constructions of this section are more efficient (especially for the DH based constructions), as described below.

The application of monotone symmetric branching programs (also called *monotone undirected contact schemes*, and *switching networks*) to secret sharing was suggested by Benaloh and Rudich [7, 28, 3] and enables to construct a secret sharing scheme for any monotone access structure (the question is the size of the shares). We first present the computational model of monotone symmetric branching programs and then a corresponding DPRF construction.

Monotone symmetric branching programs: Let $G=(V,E)$ be an *undirected* graph, $\psi : E \mapsto \{1,\ldots,n\}$ be a labeling of the edges, and s,t be two special vertices in V. A monotone symmetric branching program is defined as a tuple $\langle G,\psi,s,t\rangle$ and has boolean output. Given an input $x=\{x_1,\ldots,x_n\}\in\{0,1\}^n$, define G_x as the graph $G_x=(V,E_x)$, where $E_x=\{e \mid e\in E, x_{\psi(e)}=1\}$. The output of the program is 1 if and only if G_x contains a path from s to t.

A DPRF construction: It is possible to construct DPRFs which are either ℓ-wise independent or weakly pseudo-random, based on monotone symmetric branching programs. A user would have to receive information from a subset of the servers whose characteristic vector corresponds to a "1" output of the monotone symmetric branching program in order to obtain the required value. We present here the ℓ-wise independent construction. Note that the corresponding DH construction is more efficient than with monotone span programs since it requires only multiplications and not exponentiations.

Initialization: A monotone symmetric branching program which realizes the required access structure is constructed. A random polynomial P_s of degree $\ell-1$ is associated with the node s. The values distributed by the system are defined as $f(h)=P_s(h)$. A random polynomial P_v of degree $\ell-1$ is associated with any other vertex v, except for the vertex t to which the polynomial $P_t\equiv 0$ is assigned. Every edge $e=(u,v)$ is associated with the polynomial $P_e=P_u-P_v$. Server S_i is given the all the polynomials associated with the edges which are mapped to i (edges e for which $\psi(e)=1$).

Reconstruction: A user which wants to obtain value $f(h)$ should contact a privileged subset of the servers. Each server S_i which is approached by the user and approves of him evaluating $f(h)$ should provide it with the values $\{P_e(h) \,|\, \psi(e) = i\}$. If the user receives information from a privileged subset it can sum the values that correspond to a path from s to t and get $P_s(h)$.

Quorum systems: A Quorum system is a collection of sets (quorums), every two of which intersect (see [38] for a discussion and some novel constructions of quorum systems with optimal load and high availability). A DPRF with an access structure in which every privileged set must contain a quorum has several advantages regarding its maintenance: for example, if a user should not be allowed to compute f it is only required to inform all the servers in a *single* quorum of this restriction, and then every privileged set of servers contains at least one server which will refuse to serve that user. DPRFs with access structures based on the *paths* quorum system [38] can be efficiently realized by the constructions we presented in this section.

Efficiency: The reconstruction of the secret in the Diffie-Hellman variant we presented here requires the user to perform multiplications. It is more efficient than the reconstruction for the monotone span programs based Diffie-Hellman scheme we present in Section 4.2, which requires the user to perform exponentiations.

General prf: Note that a direct use of pseudo-random functions instead of the polynomials or of the Diffie-Hellman construction is insecure. The reason is that an edge (u, v) is associated with a function $f_u - f_v$ and since there is no concise representation for this function which hides f_u and f_v the server which is mapped to the edge should get both functions f_u and f_v. Subsequently, the server can compute $f_u(x)$ or $f_v(x)$ and not just $f_u(x) - f_v(x)$. Therefore a server which is mapped to an edge which touches s has the ability to compute by itself the value of the shared function.

4.2 Using Monotone Span Programs

It is possible to construct DPRFs with access structures which are realized by monotone span programs. Monotone span programs (MSPs) were introduced by Karchmer and Wigderson [28] and their corresponding secret sharing schemes are equivalent to linear secret sharing schemes in the sense that any secret sharing scheme in one of these classes can be realized by a scheme of the same size in the other class, see [3] for details. Recently MSPs were used by Cramer, Damgard, and Maurer [16] to construct multi-party computation protocols for general monotone sets of subsets of players, any one of which may consist of cheaters. We first present the computational model of monotone span programs and then a DPRF construction.

Monotone span programs: A monotone span program is defined by a triple $\langle K, M, \psi \rangle$ as follows. Let K be a finite field and let M be a matrix with d rows and e columns, and entries in K. The rows of M are labeled by a mapping to server identities, $\psi : \{1, \ldots, d\} \mapsto \{1, \ldots, n\}$. For a subset $A \subset \{1, \ldots, n\}$, define

M_A as the matrix consisting of the rows of M which are labeled with $i \in A$, and let d_A be the number of rows in this matrix.

Let $\epsilon = (1, 0, \ldots, 0) \in K^e$ be the *target* vector (ϵ can be replaced by any non-zero vector in K^e). An MSP computes a boolean function $f : \{0, \ldots 1\}^n \mapsto \{0, 1\}$ defined by "$f(x_1, \ldots, x_n) = 1$ if and only if ϵ is in the Image of M_A^t, where $A = \{i | x_i = 1\}$". That is, if there is a linear combination of the d_A rows labeled with an i for which $x_i = 1$, that equals the target vector ϵ. It is known that any monotone boolean function can be computed by an MSP (and the question is what size).

A DPRF construction: The construction is based on the MSP secret sharing scheme. We can achieve either ℓ-wise independence or weak pseudo-randomness. A user would have to receive information from a subset of the servers which corresponds to a "1" output of the MSP in order to obtain the required value. Following we present the DH based construction.

Initialization: An MSP which realizes the required access structure is constructed. All operations are performed over an appropriate field. A vector of random values $\bar{\alpha} = \{\alpha_1, \ldots, \alpha_e\}$ is associated with the columns of M. The function computed by the system is defined as $f(h) = h^{\alpha_1}$.

Server S_i is given the share $\bar{s}_i = M_{\{i\}}\bar{\alpha}$, which is a vector of length $d_{\{i\}}$, the number of rows in $M_{\{i\}}$.

Reconstruction: A user which wants to compute $f(h)$ should contact a privileged subset of the servers. Each server S_i which is approached by the user and approves of computing $f(h)$ should provide him with the values $\{h^\beta | \beta \in \bar{s}_i\}$ (i.e. h raised to the power of each of the coordinates of $\bar{s}_i$). . If the user receives information from a privileged subset then there is a linear combination in the exponents which obtains $f(h) = h^{\alpha_1}$. The user can perform exponentiations and multiplications to compute this combination.

5 Proactive Security

Proactive security enables a system of servers to automatically recover from repeated break-ins while preserving its security. The servers perform a periodical mutual refreshment of their secrets, and security is preserved as long as not too many servers are broken into between two refreshments (see [11] for a survey of proactive security). We can amend our schemes with proactive security while preserving consistency. The value of $f(x)$ computed in two different requests would still be the same, even if several refreshment phases pass between the two requests.

The periodic refreshment requires communication between the servers, which is a new requirement for DPRFs. Alternatively, the refreshment can be controlled by a single secure server which is the only party sending refreshment information to servers. The system is kept secure as long as there is no break-in to this server, but since this server can be highly guarded (e.g. kept off-line at all times except for refreshment phases) this scenario seems reasonable.

We describe very briefly how proactive security is obtained. The periodic refreshment phases employ techniques which are common in proactive refreshments, and a novel method for verifying that the refreshment values sent by each server are indeed correct. In the refreshment of the ℓ-wise independent construction, k servers $S_1, \ldots, S_k$ should each generate a random bivariate polynomial $P_i^t(x,y)$, subject to the constraint $P_i^t(0,\cdot) = 0$. Server S_i sends to each other server S_j the restriction of its polynomial to $x = S_j$, i.e. $P_i^t(S_j,\cdot)$. The new polynomial of each server is the sum of its old polynomial with all the new polynomials it receives.

The servers should run a verification protocol for the values they receive in the refreshment phase, in order to verify that $S_1, \ldots, S_k$ send shares of polynomials of the right degrees which are 0 for $x = 0$. This is essentially a verifiable secret sharing (VSS) protocol. It is possible to use a VSS protocol which is very efficient in both its computation and communication requirements. Very briefly, the verification is done by choosing a random point c, and requiring each S_i to broadcast $P_i^t(\cdot,c)$. Each server should verify that $P_i^t(0,c) = 0$ and that the share it received agrees with this broadcast. Note that unlike the verification protocols of [6, 20] this protocol does not require communication between each pair of servers. The random point c can be chosen in a very natural way, it can be defined as a value of the previous polynomial at a point which is only evaluated after the servers send the refreshment values.

Application to distributed initialization: The initialization of the system can be performed in a distributed manner. It is then required to verify that servers that participate in this process do not send incorrect data which would disrupt the operation of the system, i.e. that they send shares of polynomials of the right degrees. This verification can be performed very efficiently using the above protocol and a broadcast channel (note that it is not required to verify that the value of the polynomial is 0 for $x = 0$). The choice of the random point should be done by a distributed protocol which generates several values, where at least one of the values is guaranteed to be random.

Future Work

The most obvious open problem is coming with a construction which has all the properties of a DPRF, i.e. of a function which is strongly pseudo-random and can be evaluated a polynomial number of times. Another interesting line of research is the design of *oblivious* DPRFs, in which the servers do not learn what is the input x for which the user wants to compute $f(x)$. Note that the oblivious polynomial evaluation protocols of [33] are probably too expensive since the number of 1-out-of-2 oblivious transfers is linear in the degree of the polynomial.

Acknowledgments

We thank the anonymous referees for their helpful comments.

References

1. Aho A. V., Hopcroft J. E. and Ullman J. D., **The Design and Analysis of Computer Algorithms**, Addison-Wesley, 1974.
2. Bach E. and Shallit J., **Algorithmic Number Theory**, vol. 1: Efficient Algorithms, The MIT Press, 1996, pp. 155-163.
3. Beimel A., "Secure schemes for secret sharing and key distribution", DSc dissertation, 1996.
4. Bellare M. and Rogaway P., "Provably secure session key distribution — the three party case", *27th Proc. ACM Symp. on Theory of Computing*, (1995), 57–66.
5. Bellare M. and Rogaway P., Random oracles are practical: A paradigm for designing efficient protocols, *Proc. 1st Conf. on Computer and Communications Security, ACM*, 1993, pp. 62-73.
6. Ben-Or M., Goldwasser S. and Wigderson A., Completeness theorems for non-cryptographic fault tolerant distributed computation, *20th Proc. ACM Symp. on Theory of Computing*, (1988), 1-9.
7. Benaloh J. and Rudich S., private communication, 1989.
8. Blaze M,, Feigenbaum J. and Naor M., "A Formal Treatment of Remotely Keyed Encryption", *Advances in Cryptology – Eurocrypt'98*, LNCS 1403, Springer-Verlag, 1998, pp. 251–265.
9. D. Boneh, "The Decision Diffie-Hellman Problem", *Proc. of the Third Algorithmic Number Theory Symp.*, LNCS Vol. 1423, Springer-Verlag, pp. 48–63, 1998.
10. Canetti R., Towards realizing random oracles: hash functions that hide all partial information, *Adv. in Cryptology - CRYPTO '97*, LNCS, Springer, 1997, pp. 455-469.
11. Canetti R., Gennaro R., Herzberg D. and Naor D., "Proactive Security: Long-term protection against break-ins", CryptoBytes.
12. R. Canetti, O. Goldreich and S. Halevi, "The random oracle model, revisited", Proc. 30th Ann. ACM Symp. on Theory of Computing, 1998, pp. 209–218.
13. Canetti R. and Goldwasser S., "An efficient threshold public-key cryptosystem secure against chosen ciphertext attack", *Advances in Cryptology – Eurocrypt '99*, Springer-Verlag, 1999.
14. Chaum D., Crepeau C. and Damgard I., Multiparty unconditionally secure protocols, *20th Proc. ACM Symp. on Theory of Computing*, (1988), 11-19.
15. Chaum D. and Van Antwerpen H., "Undeniable signatures", *Advances in Cryptology – Crypto '89*, Springer-Verlag LNCS 435, (1990), 212–217.
16. Cramer R., Damgard I. and Maurer U., "General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme", manuscript, 1999.
17. De Santis A., Desmedt Y., Frankel Y., and Yung M., "How to share a function securely", in *Proc. 26th STOC*, 1994, pp. 522-533
18. Desmedt Y. and Frankel Y., "Threshold cryptosystems", *Advances in Cryptology – Crypto '89*, Springer-Verlag LNCS 435, (1989), 307–315.
19. Dolev D., Dwork C., Waarts O. and Yung M., "Perfectly secure message transmission", *JACM* 40(1):17–47, 1993.
20. Feldman P. and S. Micali, "An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement", Siam J. Comp. 26, 1997, pp. 873–933.
21. Gemmell P., "An introduction to threshold cryptography", *CryptoBytes*, Winter 97, 7–12, 1997.
22. Gennaro R., Jarecki S., Krawczyk H. and Rabin T., "Robust threshold DSS signatures", *Adv. in Cryptology – Eurocrypt '96*, Springer-Verlag, LNCS 1070, 354–371.

23. Goldreich O., "Foundations of Cryptography" (fragments of a book), 1995.
24. Goldreich O., Goldwasser S. and Micali S., "How to construct random functions", *J. of the ACM*, Vol. 33, No. 4, (1986), 691–729.
25. Goldreich O., Micali S. and Wigderson A., "How to play any mental game", *19th Proc. ACM Symp. on Theory of Computing*, (1987), 218-229.
26. Gong L., "Increasing availability and security of an authentication service", *IEEE J. SAC*, Vol. 11, No. 5, (1993), 657–662.
27. Kaufmaan C., Perlman R., and Speciner M., **Network Security**, Prentice Hall, 1995.
28. Karchmer M. and Wigderson A., "On span programs", *Proc. 8th Structures in Complexity Theory conf.*, 102–111, 1993.
29. Luby M., **Pseudorandomness and Cryptographic Applications**, Princeton University Press, 1996.
30. McEliece R. J. and Sarwate D. V., "On sharing secrets and Reed-Solomon Codes", *Comm. ACM*, Vol. 24, No. 9, 1981, 583–584.
31. Micali S. and Sidney R., "A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems", *Adv. in Cryptology – Crypto '95*, Springer, 185–196.
32. Naor M. and Pinkas B., Secure and efficient metering, *Advances in Cryptology – Eurocrypt '98*, LNCS 1403, Springer-Verlag, 1998, pp. 576–590.
33. M. Naor and B. Pinkas, "Oblivious Transfer and Polynomial Evaluation", *Proc. 31th Ann. ACM Symp. on Theory of Computing*, 1998.
34. Naor M. and Pinkas B., Maintaining secure communication in networks for long terms, manuscript, 1999.
35. Naor M. and O. Reingold, "Synthesizers and their application to the parallel construction of pseudo-random functions", *Proc. 36th IEEE Symp. on Foundations of Computer Science*, 1995, pp. 170-181.
36. Naor M. and Reingold O., "Number-Theoretic constructions of efficient pseudo-random functions", *Proc. 38th Symp. on Foundations of Computer Science* , 1997.
37. Naor M. and O. Reingold, "From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs", *Advances in Cryptology - CRYPTO '98*, 1998, pp. 267-282.
38. Naor M. and Wool A., "The load, capacity, and availability of quorum systems", *SIAM J. Comput.*, Vol. 27, No. 2, 423-447, April 1998.
39. Naor M. and Wool A., "Access Control and Signatures via Quorum Secret Sharing", *3rd ACM Conf. of Computer and Communication Security*, 1996, 157–168.
40. Needham R. and Schroeder M., "Using encryption for authentication in large networks of computers", *Comm. ACM*, Vol. 21, No. 12, 993–999, December 1978.
41. Rabin M. O., "Transaction Protection by Beacons", *JCSS*, Vol. 27, No. 2, (1983), 256-267.
42. Rabin T., "A Simplified Approach to Threshold and Proactive RSA", *Advances in Cryptology - CRYPTO '98*, 1998, pp. 89–104.
43. Schnorr C. P., Efficient identification and signatures for smart cards, *Proc. Advances in Cryptology - CRYPTO '89*, LNCS, Springer-Verlag, 1990, pp. 239-252.
44. Shamir A., "How to share a secret", *Comm. ACM* Vol. 22, No. 11 (Nov. 1979), 612–613.

This article was processed using the LaTeX macro package with LLNCS style