Access Control and Signatures via Quorum Secret Sharing*

Moni Naor[†‡] Avishai Wool[†§]

May 16, 1996

Abstract

We suggest a method of controlling the access to a secure database via quorum systems. A *quorum system* is a collection of sets (quorums) every two of which have a nonempty intersection. Quorum systems have been used for a number of applications in the area of distributed systems.

We propose a separation between *access servers* which are protected and trustworthy, but may be outdated, and the *data servers* which may all be compromised. The main paradigm is that only the servers in a complete quorum can collectively grant (or revoke) access permission. The method we suggest ensures that after authorization is revoked, a cheating user Alice will not be able to access the data even if many access servers still consider her authorized, and even if the complete raw database is available to her. The method has a low overhead in terms of communication and computation. It can also be converted into a distributed system for issuing secure signatures.

Keywords: Quorum systems, replication, distributed computing, security, cryptography.

---

*An extended abstract of this paper appeared in the 3rd ACM Conf. Computer and Communication Security, 1996

†Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel.

‡E-mail: `naor@wisdom.weizmann.ac.il`. Incumbent of the Morris and Rose Goldman Career Development Chair, supported by a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences and Humanities and BSF grant 94-00032.

§E-mail: `yash@wisdom.weizmann.ac.il`.

1 Introduction

1.1 Motivation

Consider the following scenario, regarding access control to a database. The satellite photography company GlobePic has a large digitized pictures database of various parts of the earth. This database is updated periodically, as new photographs are added. GlobePic customers buy the license to access a set of photographs, say of some geographic area, and this license is limited in time. When the license expires, the customer is not allowed access any more. Furthermore, the company would like to be able to quickly *revoke* the customers privileges at any time due to, say, unauthorized transfer of information. The company needs a distributed protocol to enforce this licensing policy. The protocol should run using a wide-spread collection of *access servers*, which may be completely separate from the actual *data servers*. A basic part of this protocol is just maintaining a consistent view of the licensing status of every customer, which is a classical question concerning replicated databases. Note that servers may be unavailable due to crashes or communication failures, so the protocol needs to overcome this and allow high availability of the service. On the other hand, the protocol should keep the load on any single server at a minimum.

The information in the database is highly sensitive so it must be protected. The protection should be against cheating users, rather than against dishonest access server personnel. The first requirement is that a user may not know any partial data relating to photographs that were not paid for, so the protection needs to be done per record. A second requirement is that after the license expires, and GlobePic takes the few necessary measures to update the access servers, the user is not allowed to access any photographs at all. A crucial point is that a typical replicated-database protocol does not update *all* the servers that the license has expired, so a cheating user may attempt to access the *outdated* servers, who still believe the license is valid, and thus obtain access to the photographs. Note that the access control needs to be enforced even when the whole raw database is available to a cheating user.

A closely related problem concerns distributed signatures. Suppose that the SolarCard credit company has an agreement with the LunarBank, which specifies a secure signature scheme. Any card-holder showing an officially signed letter of credit from SolarCard is entitled to receive from LunarBank the amount requested in the letter, and the credit card company is bound to reimburse the bank of the sum. Now SolarCard wishes to create an automatic signing service for its customers, via some signature servers.

A common approach to such a problem is that the user must obtain partial signatures from k of the n servers, so that fewer than k servers cannot forge a complete signature. This approach ensures both a high availability, and minimal trust in the servers. We choose to separate these issues, by trusting the servers to a large extent. However, as in the previous scenario, we assume that the servers may not all have the updated status of each customer. We argue that an important issue here is *consistency:* if the request is valid then it should be signed, but if it is not then no matter which servers the user applies to, the request must not be signed. Note that k-of-n signature schemes with $k < n/2$ do not guarantee this type of consistency even when all the servers are not corrupt. And of course a basic requirement is that cheating user should not be able to forge a signature even after receiving several partial signatures.

A solution to the problems of both scenarios includes the idea of a *quorum system*. This is a collection of sets (called quorums) every two of which have a non-empty intersection. Quorum systems are a standard tool used to maintain consistency in replicated databases. A natural formulation for the access control problem is that access to the data may be granted only by a quorum of servers. Then the intersection property guarantees that notifying all the servers in some quorum that a license has expired would make the license useless. Similarly in the signature problem, we say that only a quorum of partial signatures can be combined into a full signature, and then the intersection property guarantees the consistency.

Our purpose in this paper is to show how to add the security requirements to some of the better quorum systems, via secret sharing. The security we guarantee is against user's actions; we assume that the servers are trustworthy, but possibly unavailable or out of date. We obtain an access control protocol that has the following properties:

- Access controlled at record (rather than database) granularity.
- Very low overhead in terms of communication and computation for gaining the access.
- Low probability of denial-of-service or of giving service to an unauthorized user.
- No on-line coordination required between the servers; i.e., a user simply gets information from the servers.

We believe that the level of trust which we assume for the servers is reasonable in applications similar to the ones sketched above. However we consider decreasing this trust, without significant degradation of the protocol's advantages, to be a major open question.

1.2 Related Work

Herlihy and Tygar [20] have suggested a scheme to protect quorum based replicated databases. However their scheme has several drawbacks: access is controlled over the whole database and not at record granularity, there is no way to revoke a user's access once it is obtained, and the scheme uses a k-of-n threshold access structure, which implies a large communication overhead and high load since the threshold must be $k > n/2$ for the structure to be a quorum system.

Reiter and Birman [41] considered a database protection scheme in a scenario which is the reverse of ours, namely against servers being corrupted (rather then the users, as is the underlining assumption in our paper). In their scheme it is the responsibility of the users to verify that the the data sent by the servers is genuine. They too rely on a k-of-n threshold scheme, and do not separate between the data servers and the access servers.

The issue of availability is addressed by Gong [18], in the context of a secure authentication service. The suggested protocol uses a k-of-n threshold scheme, however in fact any secret sharing scheme could be used instead. Therefore if we replace the k-of-n scheme by our quorum secret sharing schemes, the protocol would then gain their efficiency, high availability and low load.

A related line of research is that of group signatures [10] and function sharing [9]: for processors to share a function F means that only a group of k processors can evaluate $F(x)$, and no information is transferred about F from the shares related to x. Hence, the issue of granularity is dealt with. However, their model of failures and security requirements are much more severe: the servers themselves may be corrupted, and no set of $< k$ corrupt servers is allowed to know any information about F. The corrupted servers are controlled by a malicious adversary, who knows all the secret information they may possess. The authors showed how to share the RSA function (i.e., modular exponentiation) over a threshold access structure. However, since their security requirements are stronger than ours, the problems they face are more difficult and therefore: (i) The solutions obtained are more complex than ours. (ii) They work only over threshold structures. (iii) It is not clear how to solve the problem of database encryption given this assumption on the faulty servers. The problem is in sharing a pseudo-random function.[1] (iv) As for signatures, their scheme is restricted to RSA and is not general. In particular it is not clear whether any of the provably secure signature schemes may be implemented.

1.3 Tools

Quorum systems: Quorum systems have been used in the study of distributed control and management problems such as *mutual exclusion* (cf. [13, 40]), *data replication protocols* (cf. [8, 19, 23]), *name servers* (cf. [34]) and *selective dissemination of information* (cf. [47]). We apply some recent constructions suggested in [1, 26, 35, 38].

Secret sharing: Secret sharing (cf. [43]) was originally suggested for threshold access structures by Shamir and Blakely [42, 4]. It was extended to arbitrary access structures in [22]. The issue of efficiency (i.e., share sizes) of such schemes has been considered in several papers (cf. [6, 5, 2]). Schemes suggested in [3] for structures represented by monotone formulas turn out to be important for our quorum systems. The most general access structures for which efficient secret sharing schemes are known is that of span programs [24]. All our schemes fall into this category. Krawczyk [25] suggested the notion of computational secret sharing which we adopt for our purposes.

Pseudo-random functions: Our constructions employ pseudo-random functions (cf. [16, 29]) for two purposes: encrypting the database and generating coin flips for the secret sharing schemes we use. As a heuristic, it is possible to replace the pseudo-random function with a private-key encryption function, such as DES.

Signatures: Digital signatures have been investigated extensively (cf. [17, 11] and the references therein). Our scheme can take any signature scheme and transform it into a distributed quorum based scheme, without altering its security. The notion of security we consider is that of existential unforgeability. However our transformation is

[1] We have recently made some progress regarding this question. See also [31].

independent of the basic signature scheme itself, so we can as easily create a quorum signature scheme out of heuristic signature schemes, such as RSA with MD5.

1.4 New Results

In this work we propose a separation between the *access servers* and the *data servers*. The data servers need not be be physically protected, since all the information on them will be encrypted. We shall assume, however, that none of the access servers has been compromised, although some of them may not be up-to-date regarding the access privileges of users. This may be due to communication failure or high load, or since quorum based data replication schemes allow data not to be updated at *all* the servers. The access servers have relatively low storage requirements, typically much less than that of the database itself.

We make use of quorum systems to enforce consistency of the access information. When a legitimate user Bob wants to access a record from the database, or to obtain a signature, he sends a request to all the servers in some quorum of access servers. Each of these servers, after checking authorization, sends back a message. In the case where Bob wants to get a record from the database, he also accesses the data servers and gets the encrypted record. No protection (other than encryption) is assumed on the data servers. Combining the messages Bob gets from the access servers yields the key to decrypting the record, or yields the desired signature. However if an unauthorized user Alice attempts the same procedure, she will get only a subset of replies (from the outdated servers). We show that this partial information will not be help her to forge the signature or to learn any information regarding the database which she did not have before.

The properties that interest us in a quorum systems are:

- Low load (and high capacity): the load of a quorum system is the fraction of the time that a member of the quorum system (server) is accessed under the best possible strategy of choosing quorums. Thus to allow many accesses to the database, we need a low load.
- High availability: we want a quorum of available servers to exist with high probability even when individual servers may fail. This ensures that privileges may be revoked, and that legitimate users may continue accessing the database (or obtaining signatures).
- Small quorum sizes: to make the communication overhead small.

Note that not all quorum systems enjoy these properties. For instance, the majority system [45], i.e., the $\frac{n+1}{2}$-of-n threshold system, has optimal availability but induces a high load and has large quorums. At the other extreme, the finite projective plane [30] has optimal load but very poor availability.

We suggest using some recent constructions of quorum systems that have optimal performance according to the above criteria. We show how to convert secret sharing schemes for access structures corresponding to these quorum systems into solutions for our problems. Our main results, specified in Theorems 3.1 and 5.1, show how the transformations work, and their security properties. As a consequence we get very efficient methods for protecting information in a database, or generating shared signatures. The work done at a server given a request is the evaluation of an encryption function at two points. Reconstruction by a user is also very efficient and involves XORing mostly. Our schemes require no coordination between the servers. Each server replies to a request based only on information it holds locally, and the consistency is guaranteed by the fact that obtaining replies from less than a quorum of servers does not leak information to the user.

Organization: the next section contains the definitions of quorum systems, their properties, and secret sharing schemes. Section 3 presents how to use a quorum secret sharing scheme to control the access to a database. Section 4 includes some variants to the basic protocol, with stronger security guarantees. Section 5 shows how a secret sharing scheme can be used to convert signature schemes into ones controlled by a quorum system. Section 6 gives efficient secret sharing schemes for various quorum systems, that may be used in the methods of Section 3 and 5. Section 7 describes two protocol variants which require less trust in the servers, and Section 8 lists some open problems.

2 Preliminaries

2.1 Quorum Systems, Availability and Load

Definition 2.1 *A* Set System $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ *is a collection of subsets* $Q_i \subseteq U$ *of a finite universe* U*. A* Quorum System *is a set system* $\mathcal{Q}$ *that has the* Intersection property: $Q \cap R \neq \emptyset$ *for all* $Q, R \in \mathcal{Q}$*.*

Alternatively, quorum systems are known as *intersecting set systems* or as *intersecting hypergraphs*. The sets of the system are called *quorums*. The number of elements in the underlying universe is denoted by $n = |U|$.

Definition 2.2 *A set system* $\mathcal{A} \subseteq 2^U$ *such that* $\emptyset \notin \mathcal{A}$ *is called an* access structure *if it is monotone increasing: if* $A \in \mathcal{A}$ *then* $B \in \mathcal{A}$ *for every* $B \supseteq A$*. If* $\mathcal{Q}$ *is a quorum system then the collection* $\mathcal{A}(\mathcal{Q}) = \{A \supseteq Q : Q \in \mathcal{Q}\}$ *is an access structure, called the quorum access structure of* $\mathcal{Q}$*.*

Definition 2.3 *A* Coterie *is a quorum system* $\mathcal{S}$ *that has the* Minimality property: *there are no* $S, R \in \mathcal{S}$*, s.t.* $S \subset R$*.*

Definition 2.4 *Let* $\mathcal{R}, \mathcal{S}$ *be coteries (over the same universe* U*). Then* $\mathcal{R}$ dominates $\mathcal{S}$*, denoted* $\mathcal{R} \succ \mathcal{S}$*, if* $\mathcal{R} \neq \mathcal{S}$ *and for each* $S \in \mathcal{S}$ *there is* $R \in \mathcal{R}$ *such that* $R \subseteq S$*. A coterie* $\mathcal{S}$ *is called* dominated *if there exists a coterie* $\mathcal{R}$ *such that* $\mathcal{R} \succ \mathcal{S}$*. If no such coterie exists then* $\mathcal{S}$ *is* non-dominated.

Lemma 2.5 [21] *Let* $\mathcal{S}$ *be a coterie and let* $\mathcal{A}(\mathcal{S})$ *be its access structure. Then* $\mathcal{S}$ *is non-dominated iff for all* $X \subseteq U$*, either* $X \in \mathcal{A}(\mathcal{S})$ *or* $U \setminus X \in \mathcal{A}(\mathcal{S})$ *(but not both).*

The *availability* of a quorum system is defined using a simple probabilistic model of the failures in the system. We assume that the elements (servers) fail independently with a fixed uniform probability p. We assume that the failures are *transient*, that the failures are *crash* failures (i.e., a failed element stops to function rather than functions incorrectly), and that they are *detectable*. Following [37] we define:

Definition 2.6 *For every quorum $Q \in \mathcal{Q}$ let $\mathcal{E}_Q$ be the event that Q is* hit, *i.e., at least one element $i \in Q$ has failed. Let $fail(\mathcal{Q})$ be the event that all the quorums $Q \in \mathcal{Q}$ were hit, i.e., $fail(\mathcal{Q}) = \bigcap_{Q \in \mathcal{Q}} \mathcal{E}_Q$. Let the system failure probability be $F_p(\mathcal{Q}) = \mathbb{P}(fail(\mathcal{Q}))$.*

The failure probability $F_p(\mathcal{Q})$ measures the un-availability of the quorum system $\mathcal{Q}$.

In order to define the load, we first need the notion of a strategy. A protocol using a quorum system occasionally needs to access quorums during its run. A strategy is a probabilistic rule that governs which quorum is chosen each time. In other words, a strategy gives the frequency of picking quorum Q_j.

Definition 2.7 *Let a quorum system $\mathcal{Q} = (Q_1, \ldots, Q_m)$ be given over a universe U. Then $w \in [0,1]^m$ is a* strategy *for $\mathcal{Q}$ if it is a probability distribution over the quorums $Q_j \in \mathcal{Q}$, i.e., $\sum_{j=1}^{m} w_j = 1$.*

For every element $i \in U$, a strategy w of picking quorums induces the frequency of accessing element i, which we call the *load* on i. The *system load*, $\mathcal{L}(\mathcal{Q})$, is the load on the *busiest* element induced by the *best* possible strategy. Formally, following [35]:

Definition 2.8 *Let a strategy w be given for a quorum system $\mathcal{Q} = (Q_1, \ldots, Q_m)$ over a universe U. For an element $i \in U$, the* load *induced by w on i is $\ell_w(i) = \sum_{Q_j \ni i} w_j$. The* load *induced by a strategy w on a quorum system $\mathcal{Q}$ is $\mathcal{L}_w(\mathcal{Q}) = \max_{i \in U} \ell_w(i)$. The* system load *on a quorum system $\mathcal{Q}$ is $\mathcal{L}(\mathcal{Q}) = \min_w \{\mathcal{L}_w(\mathcal{Q})\}$, where the minimum is taken over all strategies w.*

2.2 Secret Sharing

Definition 2.9 *Let S be a finite set of secrets. We say that a* secret-sharing scheme *(SSS) Π realizes an access structure $\mathcal{A}$ if Π is a mapping $\Pi : S \times R \mapsto S_1 \times \cdots \times S_n$, from the cross product of secrets and random strings to a set of n-tuples (shares) such that the following two requirements hold:*

1. *The secret can be reconstructed by any subset in $\mathcal{A}$. That is, for every secret $s \in S$ and set $A \in \mathcal{A}$ ($A = \{i_1, \ldots, i_{|A|}\}$) there exists a function $h_A : S_{i_1} \times \cdots \times S_{i_{|A|}} \mapsto S$ such that for every random string r, if $\Pi(s,r) = \{s_1, \ldots, s_n\}$ then $h_A(\{s_i\}_{i \in A}) = s$.*

2. *Every subset not in $\mathcal{A}$ can not reveal any partial information about the secret (in the information theoretic sense). Formally, for any subset $Z \notin \mathcal{A}$, for every two secrets $a, b \in S$, and for every possible collection of shares $\{s_i\}_{i \in Z}$:*

$$\mathbb{P}\left(\{s_i\}_{i \in Z} | a\right) = \mathbb{P}\left(\{s_i\}_{i \in Z} | b\right),$$

   where the probability is taken over the random string r.

We denote the share of element i by $\Pi_i(s,r)$.

Let $|x|$ denote the bit length of a value x. For a secret s we cannot expect the length of the shares to be $|\Pi_i(s,r)| < |s|$, hence the total length of the shares is at least $n|s|$. The following definition lets us measure the deviation of an SSS from the ideal space requirement.

Definition 2.10 *The* blowup factor *of an SSS* Π *is* $\beta(\Pi) = \frac{\sum_{i=1}^{n} |\Pi_i(s,r)|}{n|s|}$.

3 Access Control via Quorum Secret Sharing

3.1 Overview

In this section we show that an SSS Π that realizes a quorum access structure $\mathcal{A}(\mathcal{Q})$ can be used to build a distributed access control mechanism for a database.

The main elements of the access control are the *access servers*, which form the universe U. Each server holds a list of all the users it knows to be currently authorized to access the database. However a server's authorization list may be *outdated*. It may be the case that user Alice has had her authorization revoked, but a set $Z \subseteq U$ of servers is still unaware of this change when Alice requests access permission. Our requirement is that if all the servers in some *quorum* $Q \in \mathcal{Q}$ are informed that Alice is no longer authorized, then no matter from which set of servers Alice chooses to request access permission, she will not be able to access the database. This requirement leads naturally to our basic paradigm:

To access the database, a user must obtain permission from a quorum of access servers.

The intersection property of a quorum system then ensures that in any set $A \in \mathcal{A}(\mathcal{Q})$ (which can collectively grant permission to Alice), at least one server is informed that the request is not legitimate.

We assume also that an access server may be unavailable (due to a crash or communication failure). We would like such failures not to prevent legitimate users from obtaining access, and not to allow non-legitimate users to break in.

A further assumption we make is that each user has a secure and authenticated channel of communication with the servers. Therefore we assume that Alice cannot masquerade as Bob and obtain access permission by this.

The protocol ACP of figure 1 shows how to transform an SSS to an access control mechanism. We use the standard notion of a pseudo-random function (PRF), cf. [16, 29].

3.2 The Notion of Security

To specify the security of a cryptographic scheme we should describe (i) the type of attack assumed, i.e., the power of the adversary, and (ii) what is meant by breaking the system, i.e., what tasks the adversary can perform as a result of the attack that it could not perform before. Usually there is some "ideal" situation that we are trying to imitate. In the case of encryption of messages, a common metaphor used for the ideal is a sealed envelope. Below we first specify the model of the ideal environment and then the model of the given environment. The notion of security shall be that for any computational challenge and every

(A1) For an index x let $y(x)$ denote the content of database item x. The encrypted database item is $D(x) = y(x) \oplus Key_\alpha(x)$, where $Key_\alpha(x)$ is a PRF with a seed α.

(A2) Let Π be an SSS that realizes an access structure $\mathcal{A}$. Each server i has a procedure to compute its share, $\Pi_i(s, r)$.

(A3) The servers have the encryption PRF $Key_\alpha(x)$, and an additional PRF $R_\gamma(x)$, with a private seed γ, used to generate pseudo-random coin flips for the SSS.

Authorization check: When server i receives a request to access data item x from a user, it checks the authorization. If the request is from an unauthorized user, the server replies "REFUSE".

Share Generation: If the request is from an authorized user Bob, the server generates $k = Key_\alpha(x)$ and a pseudo-random string $r = R_\gamma(x)$. Server i then computes its share of the key, $s_i = \Pi_i(k, r)$ using the SSS, and sends s_i to Bob.

Reconstruction: When Bob collects the shares s_i from a set of servers $A \in \mathcal{A}$, he obtains the key using the reconstruction function for the set A, $k = h_A(\{s_i\}_{i \in A})$. With the key he can decrypt $y(x) = D(x) \oplus k$.

Figure 1: The access control protocol ACP.

user operating in the latter environment, there is a user in the ideal environment that has the same probability of succeeding in the challenge. This is an adaptation of semantic security for the scenario considered in this paper.

In our scenario, the ideal situation is when a user Alice$'$ interacts with the database as a black box, requesting and obtaining information about records in the database. Alice$'$ gets no information (concealed or other) about records she does not specifically request. At some point her access privileges are revoked and she gets no more information about the database. Her challenge is to compute some function G of the contents of the database (including both the parts that were revealed to her and those that were concealed). Depending on the distribution of the database, her requests, and the function G, she has a certain probability of answering this challenge correctly.

Consider now a user Alice using protocol ACP of Figure 1. We assume that her computational power is that of a probabilistic Turing machine whose running time is polynomial (in some security parameter). Alice has the complete encrypted database D as input from the data servers. Alice first performs t authorized rounds of:

- She adaptively chooses a data index x_j, and a set $A_j \in \mathcal{A}(\mathcal{Q})$ containing a quorum (based on the entire history). She requests access to item x_j from all the servers in A_j according to ACP.

- The servers $i \in A_j$ generate the shares and send them to Alice, i.e., she can then decrypt $D(x_j)$.

Then her authorization is revoked, so she performs ℓ further rounds of the same type, however now she chooses the set $Z_j \notin \mathcal{A}(\mathcal{Q})$ (since in every quorum some server will refuse her request). Again all the servers in Z_j send her their shares of the requested item x_j.

Let G be a function on $t+\ell$ variables. At the end of the $t+\ell$ rounds, Alice computes a guess g of the value $G(y(x_1), \ldots, y(x_{t+\ell}))$, based on the database contents of the indices she requested.

Our notion of security is: *for any function G (computable in probabilistic polynomial time) and every* Alice *operating in the mode described above, there is a (probabilistic polynomial time)* Alice$'$ *operating in the ideal black box model such that the difference between the probability that* Alice *computes G correctly and the probability that* Alice$'$ *computes G correctly is negligible.*

3.3 The Theorem

Theorem 3.1 *If the access structure $\mathcal{A}$ is a quorum access structure $\mathcal{A}(\mathcal{Q})$ then* ACP *has the following properties:*

1. *If all the servers in a live quorum are informed that Bob is authorized, then he can access item x.*
2. *If a quorum of servers is informed that Alice is not authorized, then she cannot learn any partial information about the database, in the sense defined above.*

Proof of property 1: If all the servers in a quorum $Q \in \mathcal{Q}$ are alive and informed of Bob's authorization, Bob can request permission from them. He will receive the shares s_i from all $i \in Q$ and by the definition of the SSS Π he would then be able to reconstruct $k = Key_\alpha(x)$ correctly and decrypt $D(x)$. ∎

Proof of property 2: We must show how to construct Alice$'$, given a description of Alice. The key point will be that if Alice and Alice$'$ have different probabilities of success in evaluating G, than we have a distinguisher between the pseudo-random functions used and truly random functions.

The machine Alice$'$ runs a simulation of Alice. To perform the simulation, Alice$'$ first gives (the simulated) Alice an "encrypted" database $D = D(1), D(2), \ldots$, by choosing completely random values for it. Then, for the first t rounds, Alice chooses x_j and $A_j \in \mathcal{A}(\mathcal{Q})$ and sends this request to Alice$'$. Alice$'$ asks the database for the contents y_j of record x_j. She then derives $k_j = D(x_j) \oplus y_j$, computes the shares $\{\Pi_i(k_j, r)\}_{i \in Z_j}$ of this key using a random string r, and provides Alice with the appropriate shares.

Then the privileges of Alice$'$ are revoked, as well the privileges of the simulated Alice. Alice$'$ simulates Alice a further ℓ rounds. For each x_j and set $Z_j \notin \mathcal{A}(\mathcal{Q})$ requested by the box Alice, Alice$'$ chooses an arbitrary value y'_j and computes $k'_j = D(x_j) \oplus y'_j$. Then Alice$'$

computes the shares $\{\Pi_i(k'_j, r)\}_{i \in Z_j}$ of this "key" using a random string r, and sends the appropriate parts to Alice. At the end of these ℓ simulated rounds, Alice outputs her guess of the function value G, which Alice$'$ outputs as the guess g'.

Suppose that the probability that Alice$'$ evaluates G correctly is significantly smaller than that of Alice. To obtain a contradiction, consider a machine Alice$''$ that operates in the same environment as Alice, modified so the functions $R_\gamma(x)$ and $Key_\alpha(x)$ of protocol ACP are replaced by truly random functions. We claim that the distribution on the communication that Alice$''$ sees and the distribution on the communication that the simulated Alice sees are identical. This is true since the encrypted database is completely random, and by the definition of the SSS Π the shares on a set $Z_j \notin \mathcal{A}(\mathcal{Q})$ are independent of the secret. Therefore the probability that Alice$'$ is correct on G is identical to the probability that Alice$''$ is correct on G.

If the difference in the probabilities of success of Alice and Alice$''$ is non-negligible, then we have a method of distinguishing the pseudo-random functions ($R_\gamma(x)$ and $Key_\alpha(x)$) from truly random functions, which violates the security assumption of PRF's. ∎

Remarks:

- For the proof we required that the database is encrypted by $D(x) = y(x) \oplus Key_\alpha(x)$ (so the simulator would be able to create a random "key" that will decode $D(x)$ into the random value y'). This forces $|Key_\alpha(x)|$ to be as long as $|y(x)|$, which could be undesirable if the value is, say, a large photograph file. It seems that using $Key_\alpha(x)$ as the (short) seed for some other PRF whose (long) output then encrypts the data would yield an equally secure protocol (this is an adaptation of Krawczyk's [25] computational secret sharing). However we do not know how to prove the security of such a modified protocol against an *adaptive* Alice which dynamically chooses which set Z_j to request shares from. (If Alice's choices are fixed in advance, then this scheme can be proved to be secure, and furthermore, each separate record is semantically secure.)

- Instead of a quorum system as in definition 2.1, we can use it's standard generalization to a read/write system, i.e., a pair of set systems $(\mathcal{R}, \mathcal{W})$ such that $R \cap W \neq \varnothing$ for any sets $R \in \mathcal{R}$ and $W \in \mathcal{W}$ (cf., [19, 12]). With this formulation, the basic paradigm is that to gain access a user must obtain permission from a read-quorum $R \in \mathcal{R}$. To revoke a user's access, a write-quorum $W \in \mathcal{W}$ must be informed. This allows more flexibility in the choice of systems, however there is a tradeoff between the availability of the read and write operations. For simplicity, we choose to concentrate on regular quorum systems.

3.4 Comparison with Alternative Solutions

Given that one desires to separate the access servers from the data servers, and given the idea of using quorums to overcome the problem of outdated servers, protocol ACP is not the only solution. Following is an alternative protocol for the access control problem (with two variants), which is more naive and eliminates the need for quorum secret sharing. Instead, the consistency is enforced by *gateways*. Each such gateway holds all the keys to the database.

In the first variant a user Bob requests the key from a gateway. Then the gateway requests authorization from the access servers on Bob's behalf (the gateway needs secure channels of communication with the access servers). If a quorum of access servers authorizes the gateway to honor Bob's request, the gateway sends the key to Bob, in one piece. This can be done since the gateway knows all the keys.

The second variant of this protocol works in the "Kerberos model" [32]. Here again there are gateways which hold all the database keys. In addition, the access servers have a signature scheme[2] which the gateways can verify. In this variant, Bob requests permission from the access servers directly. Each access server that knows Bob to be authorized replies with a signed permission token which Bob collects. Bob then sends all the collected tokens to a gateway. The gateway sends the requested key to Bob after it verifies the signatures on the tokens, and ensures that Bob indeed collected permissions from a quorum of access servers.

In order not to impair the availability and load, many gateways must be used. For example, $\Omega(\sqrt{n})$ gateways are required to obtain an availability and a load roughly equivalent to those of the Paths quorum system of Section 6.1.2. Therefore we see that these alternative solutions require essentially the same amount of trust as ACP protocol, i.e., many servers that hold all the keys.

In terms of the computations required, the gateway needs to verify $\Omega(\sqrt{n})$ signatures, which could be a costly computation, especially if the permission tokens are signed using public-key signatures. Instead, the efficient QSS schemes of Section 6 require the servers to perform only a few XOR operations for share generation. The key reconstruction, which is equally efficient, is performed separately by each user, thus it does not restrict the capacity of the system.

Moreover, both gateway-based variants require an extra round of communication, coordinated either by the gateway or by the user. Finally, the introduction of a another type of servers (the gateways) to the system increases its complexity.

We conclude that both variants of this seemingly simple protocol, are in fact inferior to our protocol ACP. Protocol ACP has a lower communication overhead, a similar (or lower) computation complexity, requires less coordination, and has less components. In fact, the alternatives are simpler than our protocol only in that they eliminate the concept of a QSS.

4 Stronger Notions of Security

4.1 Protecting Against Collaborative Attacks

The security guaranteed by Theorem 3.1 is against an attack by a *single* user. However, as presented, the protocol is vulnerable to an attack by a set of two (or more) unauthorized collaborating users, as follows. Let V be a set of users unauthorized to access item x. If each user $v \in V$ obtains the shares from a different set of servers Z_v (none of which contains

[2] A private key based authentication scheme can be used as well.

a quorum), it may be the case that the set $\cup_{v \in V} Z_v$ *does* contain a quorum, so by pooling their information, the users in V can access the database item x.

To protect the data against such an attack, we use a slightly modified protocol, ACP$'$, in which the shares are generated depending both on the requested item x and the requesting user's ID. Figure 2 contains the description of the modified share generation. All the other parts of the protocol remain the same, and there are no additional costs in communication or time.

(A4) Each user v has a unique identifier, denoted by $ID(v)$.

Share Generation: If the request is from an authorized user Bob, the server generates $k = Key_\alpha(x)$ and a pseudo-random string $r = R_\gamma(x \oplus ID(\text{Bob}))$. Server i then computes its share of the key, $s_i = \Pi_i(k, r)$ using the SSS, and sends s_i to Bob.

Figure 2: Protocol ACP$'$. Steps not shown are as in protocol ACP.

Theorem 4.1 *If the access structure $\mathcal{A}$ is a quorum access structure $\mathcal{A}(\mathcal{Q})$ then* ACP$'$ *has the following properties:*

1. *If all the servers in a live quorum are informed that Bob is authorized, then he can access item x.*

2. *For any set V of users, if for every user $v \in V$ some quorum of servers Q_v is informed that v is not authorized, then V collectively cannot learn any partial information about the database, in the sense defined in Section 3.2.*

Proof Sketch: The only difference from the proof of Theorem 3.1 is noting that a set of shares $\{s_i^v\}_{i \in Z_v}$ generated by ACP$'$ for any user $v \neq$ Alice and any set of servers Z_v not containing a quorum, can be indistinguishably generated by the simulating machine Alice$'$. ∎

4.2 Ensuring that the Shares Expire

In many applications it is undesirable to let users accumulate shares over a long period of time. As an example, consider how the accounting for such a database could work. A user should be billed for every key she obtains, i.e., for every data item x for which she received shares from a quorum of servers. Therefore, every month (say) the logs from all the servers need to be collected and tallied. If the logs are then deleted, then an authorized user Alice can avoid payment by choosing some quorum $Q \in \mathcal{Q}$, requesting one share for item x from some server $i \in Q$ in January and requesting the rest of shares from $Q \setminus \{i\}$ in February. By this she will not be logged as receiving a quorum of shares during any billing month.

The solution to this problem is to generate the shares depending on the item x and on the current *time*, in much the same way as the user-ID is used in protocol ACP$'$. Note that the servers must all use the *same* time value otherwise the key cannot be reconstructed. However, maintaining synchronized clocks is a non-trivial and costly task (cf., [44]) which we prefer to avoid. Instead, in our modified protocol ACP$''$ (see Figure 3) the user attaches a timestamp t to the request for item x, which is then used by the servers in the share generation. This might allow a cheating user to use fake timestamps, so on receiving such a request, the servers verify that the timestamp is "reasonable" before using it. For this purpose we assume that the maximal drift between any legitimate user's clock and a server's clock or between any two servers' clocks is at most Δ (which includes the time delays caused by the communication network). This Δ may be a fairly large value (e.g., 3 hours), as long as it is significantly smaller than the time period between accounting log deletions.

(A5) Each server i has a local clock T_i, and they all have the maximal clock drift value Δ.

Request: Bob sends the tuple (t, x) to some quorum of servers, where t is Bob's local clock value and x the requested data item.

Authorization check: When server i receives a request (t, x) from a user, it checks the user's authorization and the timestamp validity. If the request is from an unauthorized user, or if $|T_i - t| > \Delta$, the server replies "REFUSE".

Share Generation: If the request is from an authorized user Bob, with a valid timestamp t, the server generates $k = Key_\alpha(x)$ and a pseudo-random string $r = R_\gamma(x \oplus t)$. Server i then computes its share of the key, $s_i = \Pi_i(k, r)$ using the SSS, and sends s_i to Bob.

Figure 3: Protocol ACP$''$. Steps not shown are as in protocol ACP.

Clearly Theorem 3.1 holds for protocol ACP$''$. The following proposition, which we state without proof, shows its additional security guarantee. The statement is in terms of an imaginary global clock that works at the rate of the slowest server's local clock (we are not assuming that all the clocks have the same rate).

Proposition 4.2 *Assume that* Alice *sends a request* (t, x) *to server* i*, using protocol* ACP$''$*, which is received at global time* τ *and found to be legitimate. Then any request* (t, x) *sent by* Alice *that is received by any server* j *at global time* $> \tau + 3\Delta$*, will be refused.*

Remark: A request by Alice for the same item x but with a different timestamp $t' \neq t$ may be honored later, if t' is legitimate. However, as in Theorem 4.1, shares generated for such a request cannot be combined with shares generated for request (t, x).

5 Signatures via Quorum Secret Sharing

In this section we show that an SSS Π that realizes a quorum access structure $\mathcal{A}(\mathcal{Q})$ can be used to build a distributed signature scheme. The model of the user interacting with the servers is identical to that of Section 3, except that there are no data servers. Let us stress

(A1) For a message m let $Sig_\eta(m)$ denote the secure signature of m using a private key η.

(A2) Let Π be an SSS that realizes an access structure $\mathcal{A}$. Each server i has a procedure to compute its share, $\Pi_i(s, r)$.

(A3) Every server knows the private key η and can generate the signature $Sig_\eta(m)$, and additionally the servers have a PRF $R_\gamma(m)$, with a private seed γ, used to generate pseudo-random coin flips for the SSS.

Authorization check: When server i receives a request to sign message m from a user, it checks if the message is legitimate for this user. If it is not then the server replies "REFUSE".

Share Generation: If the message from Bob is legitimate, the server generates the signature $s = Sig_\eta(m)$ and a pseudo-random string $r = R_\gamma(m)$. It then computes its share of the signature, $s_i = \Pi_i(s, r)$ using the SSS, and sends s_i to Bob.

Reconstruction: When Bob collects the shares s_i from a set of servers $A \in \mathcal{A}$, he obtains the signature using the reconstruction function for set A, $s = h_A(\{s_i\}_{i \in A})$.

Figure 4: The quorum signature scheme QSig.

that unlike [10], the faults we allow in the servers are all *benign*, and therefore our scheme has a high level of trust in them. A server in our scheme that "turns traitor" can generate a complete signature for any message.

There are *signature servers*, which form the universe U. Each server holds a list of all the users it knows to be currently authorized to get signatures, and perhaps some more information, regarding the types of messages which should be signed. Again, it may be the case that a server's authorization list is *outdated*. Our requirement is that if all the servers in some *quorum* $Q \in \mathcal{Q}$ are informed that Alice is no longer authorized to get a signature on message m, then no matter from which set of servers Alice chooses to request a signature, she will not be able to obtain or forge it. This requirement leads naturally to our basic paradigm:

To obtain a signature, a user must obtain permission from a quorum of servers.

The intersection property of a quorum system then ensures that in any set $A \in \mathcal{A}(\mathcal{Q})$ (which can collectively sign the message m), at least one server is informed that the request is not legitimate.

Our protocol can be based on any signature scheme, without altering the security properties of the scheme. Specifically, consider the "existentially unforgeable against adaptive chosen message attacks" definition of security, as defined by [17]. Let $Sig_\eta(m)$ be a signature scheme obeying this requirement[3]. Suppose that a user Alice was authorized to get signatures for a while and then this authorization was revoked. Alice should not be able to generate a signature on *any* message for which it has not received a signature prior to revocation.

The idea of the distributed signature protocol is to imagine a database where at location m the value $Sig_\eta(m)$ is stored and apply protocol ACP to that virtual database. The protocol QSig of figure 4 shows in detail how to transform an SSS into a distributed signature scheme.

Theorem 5.1 *If the access structure $\mathcal{A}$ is a quorum access structure $\mathcal{A}(\mathcal{Q})$ then* QSig *has the following properties:*

1. *If all the servers in a live quorum are informed that Bob's message m is legitimate, then he can get a signature on m.*
2. *If a quorum of servers is informed that Alice's message m' is not legitimate, then she cannot generate a signature for m', unless she obtained the signature legally beforehand.*

The proof is very similar to the proof of Theorem 3.1.

6 Efficient Quorum Secret Sharing Schemes

In this section we show how to build quorum secret sharing schemes from several known quorum systems, with different availability and load properties. All the schemes we present are extremely efficient, with blowup factors of at most 2 and linear time complexities both for the share generation and secret reconstruction operations.

6.1 The Paths System

6.1.1 The System

The Paths system [35] is based on paths in the following grid.

Definition 6.1 *Let $G(d)$ be the planar grid with vertex set $\{(v_1, v_2) : 0 \le v_1 \le d+1, 0 \le v_2 \le d\}$ and edge set consisting of all edges joining neighboring vertices* except *those joining vertices u, v with either $u_1 = v_1 = 0$ or $u_1 = v_1 = d+1$. Let $G^*(d)$, be the dual of $G(d)$ with vertex set $\{v + (\frac{1}{2}, \frac{1}{2}) : 0 \le v_1 \le d, -1 \le v_2 \le d\}$ and edge set consisting of all edges joining neighboring vertices* except *those joining vertices u, v with either $u_2 = v_2 = -\frac{1}{2}$ or $u_2 = v_2 = d + \frac{1}{2}$.*

See Figure 5 for a drawing of $G(d)$ and $G^*(d)$. Note that every edge $e \in G(d)$ has a dual edge $e^* \in G^*(d)$ which *crosses* it. We call such e and e^* a *dual pair of edges.*

[3] Note that using [14] we can convert any signature scheme into one where each message has a unique signature.

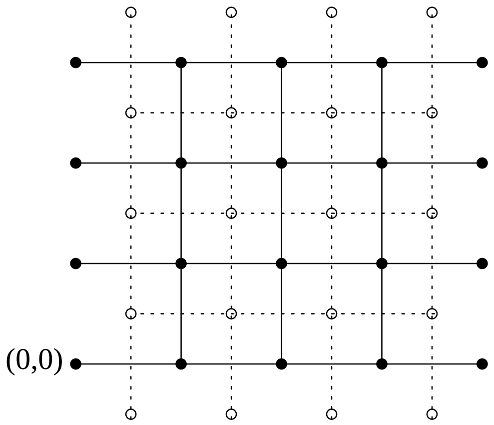

Figure 5: The grids $G(3)$ (full lines) and $G^*(3)$ (dotted lines).

Definition 6.2 *We identify an element in the* Paths *quorum system with a dual pair of edges $e \in G(d)$ and $e^* \in G^*(d)$. A quorum in the system is the union of (elements identified with) the edges of a left-right path in $G(d)$ and the edges of a top-bottom path in $G^*(d)$.*

In [35] it is shown that the Paths system has an optimal load of $\mathcal{L}(\text{Paths}) = O(1/\sqrt{n})$. It also achieves the highest availability possible for such a load, namely $F_p(\text{Paths}) = \exp(-\Omega(\sqrt{n}))$. The smallest quorums in the Paths system have cardinality $O(\sqrt{n})$.

6.1.2 The Scheme Paths-SSS

The scheme is based on the construction of Rudich for $s-t$ connectivity that was generalized in [24] for span programs. The system elements in the Paths system are the *edges* of the grid, however we first assign intermediate values to the vertices, from which we compute the shares.

The basic secret unit s is a single bit. The secret is first randomly split into four bits l, r, t and b, such that $l \oplus r \oplus t \oplus b = s$.

We describe how the l and r bits are shared by the left-right paths. Every vertex v is assigned a bit x_v. The vertices on the left side of the grid all get $x_v \leftarrow l$, and those on the right side get $x_v \leftarrow r$. For every other v the bit x_v is chosen independently at random. An edge $e = (u, v)$ is assigned a left-right share bit, $s_e^{lr} = x_v \oplus x_u$.

This procedure is now repeated on the dual grid, with bits t and b assigned to the top and bottom vertices, resp., and random values assigned to all the other vertices. The dual edges e^* are assigned shares $s_{e^*}^{tb}$ analogously. Finally, the share of a system element identified with the dual pair of edges e, e^* is both the s_e^{lr} and s_e^{tb} bits.

Proposition 6.3 *Paths-SSS is a secret sharing scheme realizing the quorum access structure* $\mathcal{A}(\text{Paths})$*, with* $\beta(\text{Paths-SSS}) = 2$.

In order to prove Proposition 6.3 we need to show the reconstruction function for a quorum, and the independence of shares of a non-quorum from the secret. We do this via the following two lemmas.

Lemma 6.4 *A quorum Q in the Paths system can reconstruct the secret bit s from the shares $\{s_e\}_{e \in Q}$ generated by Paths-SSS.*

Proof: By the definition of the quorums, there exists a set $F \subseteq Q$ such that the edges $e \in F$ form a left-right path, and a set $H \subseteq Q$ forming a top-bottom path on the dual grid. We claim that

$$s = \left(\bigoplus_{e \in F} s_e^{lr} \right) \oplus \left(\bigoplus_{e^* \in H} s_{e^*}^{tb} \right).$$

This is since in the first XOR the random values assigned to intermediate vertices in $G(d)$ cancel out and only $l \oplus r$ remains, and similarly for $t \oplus b$ on the dual grid. ∎

Lemma 6.5 *A set Z not containing a quorum can reveal no partial information of the secret s from the Paths-SSS shares $\{s_e\}_{e \in Z}$.*

Proof: Assume w.l.o.g. that Z contains a top-bottom path, i.e., the value $t \oplus b$ is known to the adversary (this only helps the adversary). Consider a set of shares $\zeta = \{s_e\}_{e \in Z}$. Since these are legitimate shares, they have an extension to a full set of shares. Specifically, there exist bits x_v and a secret $l \oplus r$ which define shares $\sigma = \{s_e\}_{e \in U}$ such that σ encodes $l \oplus r$ by the Paths-SSS scheme, and $\sigma|_Z = \zeta$. We show that for every such σ there exists a different set of shares σ' which encodes the secret $\overline{l \oplus r}$, and for which $\sigma'|_Z = \zeta$ as well.

Since Z does not have a quorum, it does not contain any left-right path. Therefore we can partition the vertices into two disjoint sets V and W such that V contains all the vertices that have a path to the left side of $G(d)$ in Z, and W contains the vertices with a path to the right. Note that there is no edge in Z connecting a vertex $v \in V$ with $w \in W$.

We now construct the corresponding shares $\sigma' = \{s'_e\}_{e \in U}$, as follows. Every vertex v is assigned a bit y_v. Set $y_v = x_v$ for all the vertices $v \in V$, and set $y_w = \bar{x}_w$ for all $w \in W$ (including the right side vertices which have $x_w = r$). The shares are computed as before: for every edge $e = (v, w)$, regardless of whether $e \in Z$ or not, set $s'_e = y_v \oplus y_w$.

Now XORing the shares s'_e on a left-right path would compute $\overline{l \oplus r}$, since we flipped the r bit. However, $\sigma'|_Z = \sigma|_Z = \zeta$. This is since the endpoints of an edge $e \in Z$ can be either both in V or both in W, so either s'_e is not touched (if both endpoints are in V), or remains the same after flipping the bits at both endpoints.

This shows that for every set of shares ζ on Z there is a 1–1 correspondence between an extension σ encoding $l \oplus r$ and an extension σ' encoding $\overline{l \oplus r}$. Hence $\mathbb{P}(\zeta | l \oplus r) = \mathbb{P}(\zeta | \overline{l \oplus r})$. ∎

6.2 The HQS and Tree Systems

6.2.1 The Systems

The hierarchical quorum system (HQS) is due to [26]. In this system the elements are the leaves of a complete ternary tree, in which the internal nodes are 2-of-3 majority gates.

In [36] the availability and load of the HQS are analyzed. It is shown that $\mathcal{L}(\text{HQS}) = n^{-0.37}$. The HQS has the highest availability possible for such load, namely $F_p(\text{HQS}) \leq$

$\exp(-\Omega(n^{0.63}))$ when $p < \frac{1}{3}$ and $F_p(\text{HQS}) \leq n^{-\alpha(p)}$ when $p < \frac{1}{2}$ (for some function α independent of n). The quorums in the HQS are all of size $n^{0.63}$.

The Tree quorum system of [1] can also be described as a ternary tree, with internal nodes which are 2-of-3 majority gates. However only two input lines of each internal majority gate are connected to lower level gates; the middle input line is directly labeled with a system element. Note that the standard description of this quorum system is via a *binary* tree; the description given here is from [21].

The smallest quorums in the Tree are of size $\log n$. In [37, 36] it is shown that the Tree has optimal availability and load among the quorum systems with logarithmic size quorums, namely $F_p(\text{Tree}) = O(n^{-\varepsilon})$ for some constant $\varepsilon(p) > 0$, and

$$\mathcal{L}(\text{Tree}) = O(\frac{1}{\log n}).$$

6.2.2 The Schemes HQS-SSS and Tree-SSS

The building block for both the Tree and HQS systems is the 2-of-3 majority gate, which is a threshold function. As such, it has an SSS, which is Shamir's scheme [42]. However a 2-of-3 majority gate is a very simple case of the general scheme. The underlying field is the $GF(4)$ field, i.e., the basic secret unit is a pair of bits. The random polynomials of the scheme are simply lines (of degree 1). Therefore both the share generation and secret reconstruction require only the few instructions in $GF(4)$ arithmetic needed for linear interpolation. Applying this scheme recursively in the natural way, from the root of the tree towards the leaves, yields secret sharing schemes for both the HQS and Tree systems.

Note that in each internal majority gate, the three shares generated from a secret value s (a pair of bits) are of two bits each. Since in both the Tree and HQS systems each element is identified with a single input line, the shares in the full HQS or Tree scheme are also of two bits each, so there is no blowup, and $\beta(\text{HQS-SSS}) = \beta(\text{Tree-SSS}) = 1$.

Remark: The above scheme works for any quorum system having a description in the form of a tree of 2-of-3 majority gates, with the system elements labeling the input lines. In [33, 21, 27] it is shown that any maximal quorum system has such a description. However no bounds are shown for the of number times that an element can appear on input lines, so using our scheme on such a description could potentially cause a high blowup.

6.3 The Crumbling Wall System

6.3.1 The System

The Crumbling Walls (CW) are a family of quorum systems due to [38]. This family includes, among others, the CWlog system (see Figure 6), the grid of [7] and the triangular wall of [28].

The elements of a wall are logically arranged in *rows* of varying *widths*. A quorum in a wall is the union of one full row and a representative from every row below the full row.

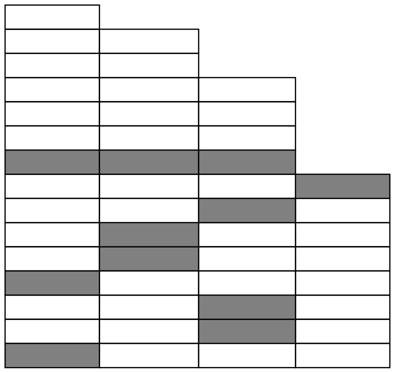

Figure 6: A CWlog with $n = 49$ elements and $d = 15$ rows, with one quorum shaded.

Here we concentrate on walls in which the top row has width $n_1 = 1$ and every other row has width $n_i \geq 2$. In [38] it is shown that such walls are non-dominated coteries (recall Definition 2.4).

In the CWlog system, the width of row i is $n_i = \lfloor \lg 2i \rfloor$. In [39] it is shown that the CWlog system is essentially the only high availability wall. It has small quorums, of size $O(\log n)$, and optimal availability and load among the quorum systems with logarithmic size quorums, namely $F_p(\mathrm{CWlog}) = O(n^{-\varepsilon})$ for some constant $\varepsilon(p) > 0$, and

$$\mathcal{L}(\mathrm{CWlog}) = O(1/\log n).$$

6.3.2 The Scheme CW-SSS

Consider a wall CW of d rows, with row 1 having width $n_1 = 1$ and $n_i \geq 2$ for all $i \geq 2$. The basic secret unit s is a single bit. This secret s is first randomly split into d bits such that $v_1 \oplus \cdots \oplus v_d = s$. Using these v_i bits we can define their partial parities, $t_i = v_1 \oplus \cdots \oplus v_{i-1}$, and $t_1 = 0$. For a row i, split t_i randomly into n_i bits h_i^j such that $h_i^1 \oplus \cdots \oplus h_i^{n_i} = t_i$. The share s_i^j of the j'th element in row i is two bits: v_i and h_i^j.

Lemma 6.6 *A quorum Q in the wall can reconstruct the secret bit s from the shares $\{s_i^j\}$ generated by CW-SSS.*

Proof: By definition the quorum Q contains a full row i and a representative in each row $k > i$. We claim that

$$s = \left(\bigoplus_{j=1}^{n_i} h_i^j \right) \oplus \left(\bigoplus_{k \geq i} v_k \right). \qquad (*)$$

The first XOR computes the value t_i of row i, which is the partial parity of v_k's above row i, so the whole expression is precisely the parity of all v_k's. ∎

Lemma 6.7 *A set Z not containing a quorum can reveal no partial information of the secret s from the CW-SSS shares.*

Proof: Let σ be a set of shares encoding the secret bit s. We show a corresponding set of shares σ' such that $\sigma|_Z = \sigma'|_Z$ but σ' encodes the bit $\bar{s}$.

As noted above, CW is a non-dominated coterie. Since $Z \notin \mathcal{A}(\mathrm{CW})$ then by Lemma 2.5 it follows that $U \setminus Z \in \mathcal{A}(\mathrm{CW})$. Therefore there exists a quorum $Q \subseteq U \setminus Z$. By definition Q contains all the elements of some row ℓ, and a representative in every row $i > \ell$.

To obtain the shares σ', we flip the v_ℓ values for all the elements in row ℓ (in the quorum Q), and flip the h_i^j values for every representative element of Q which is in row $i > \ell$. This procedure generates correct shares for the secret $\bar{s}$, since:

- A quorum R based on a full row $i \leq \ell$ necessarily contains an element of row ℓ (or possibly the whole row ℓ). So the flipped value v_ℓ enters the computation in the second XOR in $(*)$ and every other value is unaffected, hence the shares of R construct the secret $\bar{s}$.
- A quorum T based on a full row $i > \ell$ contains a representative element of Q in row i, say element j in this row. Then the flipped value of this h_i^j will appear in the first XOR in $(*)$, and again the shares will reconstruct $\bar{s}$.

However note that the shares on the set Z in σ are identical to those in σ', since changes were only made at elements of the quorum Q, which is disjoint from Z. Therefore for every set of shares ζ on Z we have shown a 1–1 correspondence between an extension σ encoding s and an extension σ' encoding $\bar{s}$. Hence $\mathbb{P}(\zeta|s) = \mathbb{P}(\zeta|\bar{s})$, and we are done. ∎

Proposition 6.8 *For any crumbling wall* CW*, CW-SSS is a secret sharing scheme realizing the quorum access structure* $\mathcal{A}(\mathrm{CW})$*, with* $\beta(\text{CW-SSS}) = 2$.

6.4 The AndOr System

6.4.1 The System

The AndOr system appears in [35], and applies the analysis of AND/OR trees of [46]. Consider a complete rooted binary tree of height h, and identify the $n = 2^h$ leaves of the tree with the system elements.

The AndOr system is the conjunction of two monotone boolean functions, defined by assigning AND and OR gates to the internal nodes of the tree, over the same inputs. The gates alternate between AND and OR, level by level. The only difference between the two functions is that one function has an AND gate at the root while the other has an OR gate there.

In [35] it is shown that this indeed is a quorum system, with quorums of size $O(\sqrt{n})$. It has an optimal load of $\mathcal{L}(\mathrm{AndOr}) = O(1/\sqrt{n})$. The availability is as high as possible, namely $F_p(\mathrm{AndOr}) \leq \exp(-\Omega(\sqrt{n}))$ when $p < \frac{1}{4}$, and $F_p \leq \exp(-\Omega(n^{0.19}))$ if $p \leq 0.38 - \Omega(n^{-0.19})$.

6.4.2 The Scheme AndOr-SSS

Since the AndOr system has a very simple description in terms of boolean functions, we can apply the scheme of [3]. The basic secret unit is a single bit. A secret s is (recursively) split over an OR gate by assigning each input a copy of s, and over an AND gate by assigning the inputs with random values s_1, s_2 such that $s_1 \oplus s_2 = s$. In [3] it is proved that this is an SSS for any monotone boolean function.

In the formula for the AndOr system each element appears on two input gates (one in each alternating tree), so each element will end with a share of two bits from the original secret bit s, hence the blowup is $\beta(\text{AndOr-SSS}) = 2$.

7 Towards Decreasing the Trust

7.1 A General Approach

First, we present a simple solution to the access control problem, in which a coalition of servers that does not contain a quorum cannot grant access to any data item x. This is a general solution, that works for any quorum access structure. The protocol relies on additional information attached to the database by the *dealer*, the all-knowing entity that prepares the database.

We assume that every access server i has a PRF E_i, and that the dealer knows all the private seeds of these PRF's. However the servers do *not* have the encryption PRF, $Key_\alpha(x)$. The protocol also uses a SSS Π for a quorum access structure $\mathcal{A} = \mathcal{A}(\mathcal{Q})$.

When the dealer writes data item x into the database, encrypted by $k = Key_\alpha(x)$, it also creates all n shares $\{s_i = \Pi_i(k, r)\}_{i=1}^n$, using a random (or private pseudo-random) string r. Then the dealer generates the values $F_i(x) = E_i(x) \oplus s_i$ using the servers' PRF's, and attaches the set of $\{F_i(x)\}_{i=1}^n$ to the data item x.

The access server's role is extremely simple. If a user Bob is authorized to access item x, server i sends back the value $E_i(x)$. Once Bob collects replies from a set of servers $A \in \mathcal{A}$ containing a quorum, he can decode the shares $s_i = F_i(x) \oplus E_i(x)$ for all $i \in A$, since the values $F_i(x)$ are available to him. Then he obtains the key using the reconstruction function for the set A, $k = h_A(\{s_i\}_{i \in A})$.

It is not hard to see that this protocol has minimal trust in the servers. In fact, any set $Z \notin \mathcal{A}(\mathcal{Q})$ of malicious servers can learn nothing about the contents of the database, in the sense of Section 3.2. However the protocol has several drawbacks:

- The size of each data item is increased by $n \cdot |Key_\alpha(x)|$ bits. This makes the protocol ridiculous if we make the assumptions of protocol ACP, namely that the key is as long as the data item itself, since the database size increases by a factor of $n + 1$. However if the key is much shorter than the data (cf. [25]), then this overhead can be tolerated.

- The protocol is not flexible, in that it requires the number of servers n and their private functions E_i to be fixed when the database is *created*. In contrast, in our approach a data item is always encrypted with the same key. Thus if the quorum system is changed there is no need to modify the data, and it suffices to update the access servers.

- The protocol cannot be used as a group signature scheme, since this would require the dealer to pre-compute and store every possible signature in advance.
- As presented, the scheme is vulnerable to the attacks mentioned in Section 4, namely collaborating users can access the data, and keys do not expire. The solutions we suggested there are not applicable, since the servers cannot generate the shares "on-the-fly" (only the dealer knows the random string r). This can be overcome by running protocol ACP$'$ (say) in parallel, using a separate key k'. The content $y(x)$ of item x would then be encrypted by $D(x) = y(x) \oplus k \oplus k'$.

7.2 The Case of the Paths System

Here we construct another access control protocol, based specifically on the scheme Paths-SSS of Section 6.1. This protocol has the same security guarantee as that of the general ACP protocol (Theorem 3.1). However it needs less trust and can tolerate some malicious servers. More precisely, there are two classes of servers: the class T of *trusted* servers ($|T| < 4\sqrt{n}$) and the class of regular servers $R = U \setminus T$. The security is compromised only if at least two trusted servers "turn traitor" (in the worst possible choice of traitors). Any number of regular servers can turn traitor without compromising the security.

Instead of the PRF $Key_\alpha(x)$ used in ACP, we have four distinguished PRF's, denoted by $K_l(x)$, $K_r(x)$, $K_t(x)$ and $K_b(x)$. The content $y(x)$ of item x is encrypted by $D(x) = y(x) \oplus K_l(x) \oplus K_r(x) \oplus K_t(x) \oplus K_b(x)$. We place a copy of K_l at each vertex on the left side of the grid $G(d)$ and a copy of K_r on the right side (see Figure 5), and similarly K_t and K_b at the top and bottom vertices of $G^*(d)$. At each internal grid vertex $v \in G(d)$ or $v \in G^*(d)$ we place a different PRF $K_v(x)$, which is unique to that vertex. Note that the PRF's placed at the internal vertices play no part in the encryption of the database.

Recall that the elements (servers) are identified with dual pairs of grid edges. Therefore a server identified with the dual edges $e = (v_1, v_2)$ and $e^* = (u_1, u_2)$ is given the four PRF's that were placed at the vertices v_1, v_2, u_1, u_2. The share that such a server generates for item x is comprised of a left-right share, $s^{lr}_e = K_{v_1}(x) \oplus K_{v_2}(x)$ and a top-bottom share, $s^{tb}_{e^*} = K_{u_1}(x) \oplus K_{u_2}(x)$.

As in Proposition 6.3, if $e_1, \ldots, e_p$ is a left-right path in $G(d)$ then $s^{lr}_{e_1} \oplus \cdots \oplus s^{lr}_{e_p} = K_l(x) \oplus K_r(x)$, and similarly on the dual grid. Therefore by collecting the shares from a quorum of servers, the user can reconstruct the key and access the data item.

A moment's reflection shows that Theorem 3.1 holds for this protocol as well. However here we have the stronger guarantee of the following proposition.

Proposition 7.1 *Let T_1, T_2, T_3, T_4 be the (non-disjoint) sets of servers whose corresponding edges touch the left, right, top and bottom sides of the grid, respectively. Then any set of traitor servers Z for which $Z \cap T_j = \varnothing$ (for some $1 \le j \le 4$) can learn nothing about the contents of the database, in the sense of Section 3.2.*

Call $T = \cup_{1 \le j \le 4} T_j$ the set of *trusted* servers, and call $R = U \setminus T$ the set of *regular* servers. Clearly $|T| < 4\sqrt{n}$. If Z is a coalition of traitor servers, then some immediate corollaries of Proposition 7.1 are:

- If $Z \subseteq R$ then Z can learn nothing of the database.
- If $|Z| = 1$ then Z can learn nothing of the database.
- If $|Z| = 2$ and Z is not a pair of diagonally opposite corners, then Z can learn nothing of the database.
- If all four corners are not traitors, and $|Z| < 4$, then Z can learn nothing of the database.

8 Open Problems

This work suggests several lines of research, which we outline below.

- Make the protocols work with less trust. Ideally, assume that a subset of the access servers are faulty and pool together all their information. Find an *efficient* scheme to protect the information so that a faulty set of the access servers that does not contain a quorum can learn nothing about the database, but from a set of the servers that includes a quorum it is easy to extract a key for decrypting any database item x[4]. Note that the availability measures the probability that the faulty processors contain a quorum system (assuming that each processor becomes faulty independently with probability p).
- We have found secret sharing schemes for many interesting quorum systems (falling into the category of span programs), however there are some for which it is not clear whether a good scheme exists, for example the projective plane [30]. Find secret sharing schemes for these quorum systems, or better, derive a general construction.
- In the remarks following the proof of Theorem 3.1 we pointed out a delicate problem: what happens when there are many large files encrypted on, say, a CD-ROM, and keys for decrypting them may be obtained for a fee. After some keys have been obtained, how secure are the remaining files? If the keys encrypting the files are chosen independently, then at first it seems obvious that nothing can be learned about the other files. However, note that the files are opened at the user's request *after* seeing their encrypted versions. To the best of our knowledge, the common definition of security of encryption (semantic, see [15, 29]) does not allow us to conclude the following: if the keys for decrypting 50 out of the 100 files (say) are given, then nothing can be learned about the remaining 50 files. Find either a way of showing that the security of the remaining files does follow from the semantic security of the encryption scheme, or find an encryption scheme for which you can prove the security with keys which are significantly shorter than the files themselves.

[4]We can show that *any* quorum system has such a scheme, but the key sizes are, in general, exponential in the number of elements.

Acknowledgments

We are very grateful to Hugo Krawczyk for his numerous comments on an early version of this paper. We thank Don Coppersmith and an anonymous referee, whose remarks allowed us to improve our presentation.

References

[1] D. Agrawal and A. El-Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comp. Sys.*, 9(1):1–20, 1991.

[2] A. Beimel and B. Chor. Universally ideal secret sharing schemes. In *Advances in Cryptology - CRYPTO'92, LNCS 740*, pages 183–195. Springer-Verlag, 1992.

[3] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Advances in Cryptology - CRYPTO'88, LNCS 403*, pages 27–36. Springer-Verlag, 1988.

[4] G. R. Blakely. Safeguarding cryptographic keys. *Proc. AFIPS, NCC*, 48:313–317, 1979.

[5] C. Blundo, A. De Santis, L. Gargano, and U. Vaccaro. On the information rate of secret sharing schemes. In *Advances in Cryptology - CRYPTO'92, LNCS 740*, pages 148–167. Springer-Verlag, 1992.

[6] E. F. Brickell and D. M. Davenport. On the classification of ideal secret sharing schemes. In *Advances in Cryptology - CRYPTO'89, LNCS 435*, pages 278–285. Springer-Verlag, 1990.

[7] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Trans. Knowledge and Data Eng.*, 4(6):582–592, 1992.

[8] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, 1985.

[9] A. De Santis, Y. Desmet, Y. Frankel, and M. Yung. How to share a function securely. In *Proc. 26th ACM Symp. Theory of Computing*, pages 522–533, 1994.

[10] Y. Desmet and Y. Frankel. Shared generation of authenticators and signatures. In *Advances in Cryptology - CRYPTO'91, LNCS 576*, pages 457–469. Springer-Verlag, 1991.

[11] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. In Yvo G. Desmedt, editor, *Advances in Cryptology - CRYPTO'94, LNCS 839*, pages 234–246. Springer-Verlag, 1994.

[12] A. Fu. *Enhancing Concurrency and Availability for Database Systems.* PhD thesis, Simon Fraser University, Burnaby, B.C., Canada, 1990.

[13] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *J. ACM*, 32(4):841–860, 1985.

[14] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In A.M. Odlyzko, editor, *Advances in Cryptology - CRYPTO'86, LNCS 263*, pages 104–110. Springer-Verlag, 1987.

[15] O. Goldreich. Foundations of cryptography (fragments of a book). Electronic Colloquium on Computational Complexity, 1995. Electronic Publication: `http://www.eccc.uni-trier.de /eccc/info/ECCC-Books/eccc-books.html`.

[16] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33:792–807, 1986.

[17] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.

[18] L. Gong. Increasing availability and security of an authentication service. *IEEE J. Selected Areas Comm.*, 11(5):657–662, 1993.

[19] M. P. Herlihy. *Replication Methods for Abstract Data Types*. PhD thesis, Massachusetts Institute of Technology, MIT/LCS/TR-319, 1984.

[20] M. P. Herlihy and J. D. Tygar. How to make replicated data secure. In *Advances in Cryptology - CRYPTO'87, LNCS 293*, pages 379–391. Springer-Verlag, 1988.

[21] T. Ibaraki and T. Kameda. A theory of coteries: Mutual exclusion in distributed systems. *IEEE Trans. Par. Dist. Sys.*, 4(7):779–794, 1993.

[22] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. IEEE Global Telecommunication Conf. (Globecomm 87)*, pages 99–102, 1987.

[23] S. Jajodia and D. Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. Database Sys.*, 15(2):230–280, 1990.

[24] M. Karchmer and A. Wigderson. On span programs. In *Proc. Structures in Complexity Theory*, pages 102–111, 1993.

[25] H. Krawczyk. Secret sharing made short. In *Advances in Cryptology - CRYPTO'93, LNCS 773*, pages 136–146. Springer-Verlag, 1994.

[26] A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Comput.*, 40(9):996–1004, 1991.

[27] D. E. Loeb. A new proof of Monjardet's median theorem, 1994. Manuscript, to appear in J. Comb. Theory, Series A.

[28] L. Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conf. Combinatorics, Graph Theory and Computing*, pages 3–12, 1973.

[29] M. Luby. *Pseudorandomenss and Cryptographic Applications.* Princeton Universtity Press, Princeton, New Jersey, 1996.

[30] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comp. Sys.*, 3(2):145–159, 1985.

[31] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions for applications to clipper-like key escrow systems. In D. Coppersmith, editor, *Advances in Cryptology - CRYPTO'95, LNCS 963*, pages 185–196. Springer-Verlag, 1995.

[32] S. P. Miller, B. C. Neuman, J. L. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan, Section E.2.1, MIT, 1988.

[33] B. Monjardet. Charactérisation des éléments ipsoduaux du treillis distributif libre. *C. R. Acad. Sc. Paris, série A*, 274:12–15, 1972.

[34] S. J. Mullender and P. M. B. Vitányi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[35] M. Naor and A. Wool. The load, capacity and availability of quorum systems. In *Proc. 35th IEEE Symp. Foundations of Comp. Sci. (FOCS)*, pages 214–225, 1994. To appear in SIAM J. Computing.

[36] M. Naor and A. Wool. The load, capacity and availability of quorum systems. Technical Report CS95-03, The Weizmann Institute of Science, Rehovot, Israel, 1995.

[37] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210–223, 1995.

[38] D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. In *Proc. 14th ACM Symp. Princip. Distributed Computing (PODC)*, pages 120–129, Ottawa, Canada, 1995. To appear in Distributed Computing.

[39] D. Peleg and A. Wool. The availability of crumbling wall quorum systems. *Discrete Applied Math.*, 1996. To appear.

[40] M. Raynal. *Algorithms for Mutual Exclusion.* MIT press, 1986.

[41] M. K. Reiter and K. P. Birman. How to securely replicate services. *ACM Trans. Prog. Lang. Sys.*, 16(3):986–1009, 1994.

[42] A. Shamir. How to share a secret. *Comm. ACM*, 22(11):612–613, 1979.

[43] G. J. Simmons. An introduction to shared secret and/or shared control schemes and their application. In *Contemporary Cryptology, The Science of Information Integrity*, pages 441–497. IEEE Press, 1992.

[44] B. Simons, J. L. Welch, and N. Lynch. An overview of clock synchronization. In B. Simons and A. Spector, editors, *Fault-Tolerant Distrib. Comp., LNCS 448*. Springer-Verlag, 1990.

[45] R. H. Thomas. A majority concensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Sys.*, 4(2):180–209, 1979.

[46] L. G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5:363–366, 1984.

[47] T. W. Yan and H. Garcia-Molina. Distributed selective dissemination of information. In *Proc. 3rd Inter. Conf. Par. Dist. Info. Sys.*, pages 89–98, 1994.