Maille Authorization — A Distributed, Redundant Authorization Protocol

Andrew Fritz and Jehan-François Pâris
Department of Computer Science
University of Houston, Houston, TX 77204-3010

afritz@uh.edu, paris@cs.uh.edu

Abstract

The Maille Authorization protocol provides flexible and reliable authorization in large distributed and pervasive computing systems. Service owners distribute their access control lists across the network using threshold cryptography. Instances of the distributed service need only verify that requestors have knowledge of a specific secret provided by the Maille Authorization system. Requestors use the Maille protocol to find and retrieve individual parts of the scattered key. Once a sufficient quorum of nodes holding the key is found, the requestor can reassemble the key and is authorized. Unlike extant systems, the Maille Authorization protocol has no single administrative point of failure and tolerates multiple simultaneous Byzantine failures.

1. Introduction

Within distributed computer systems, services are the fundamental resource. Authorization protects both simple data (files, etc) and more complex services from unauthorized actions by users. Authorization is the process of making sure a requestor is allowed to carry out a given action and relies explicitly on the ability to determine the name of the user in question (authentication). Authorization is important because unauthorized accesses may have legal ramifications, cause lost profits, or result in loss of work.

In small systems, authorization is handled in an ad-hoc manner. Service owners implicitly trust the computer systems hosting their service. The computer system verifies the requestor's identity (it authenticates them) and then consults the access policy to determine if the request can be allowed to complete. In some distributed systems, a similar scheme is used. Individual computer systems store the access policies for the services they host. Service owners must still trust each system that replicates their service.

This all-in-one approach fails when the services are scattered over a large set of untrusted hosts. In such systems, many computers may hold partial or complete copies of the service. Further, one requestor may use several replicas at once to increase throughput. As a result, widely distributed services need to decouple authorization from hosting. Plutus [8] achieves this goal by (a) using encryption and (b) letting the client handle all the key management and distribution. Woo and Lam [13] assign all authorization task to one or more specialized servers.

Both approaches have significant drawbacks. Delegating all key management and distribution to the clients just shifts the problem. Relying on a few trusted authentication servers to control access to services assumes that we have such hosts and that they cannot be compromised. This is a dubious assumption in large distributed systems. In addition, the fact that any large-scale authorization system will have to be replicated increases the chance that at least one of the authorization servers could be compromised.

We propose a more robust and more scalable solution. The Maille Authorization protocol scatters all authorization tasks among subsets of nodes. It relies on threshold cryptography [9] to ensure that it will continue to operate correctly in the presence of a small number of simultaneous Byzantine failures. In addition, the protocol degrades gracefully in the presence of increasing numbers of Byzantine failures. A catastrophic failure, comparable to the theft of a Kerberos password file [10], could only occur if nearly all nodes in the network are compromised.

To achieve these goals, the Maille Authorization protocol assigns one unique access key to each type of access on each service. This access key is broken into parts using threshold cryptography. Each part of the key is then paired with a copy of the access list for that key to form an access packet. The access packets for a service are then scattered to random member nodes in a peer-to-peer network.

When users want to use a service, they initiate a search for access packets of that service and wait for them to be returned. When a node holding an access packet receives a query, it checks to see if the requestor is listed in the authorized list and then authenticates the requestor. If the requestor passes these tests, the node returns a copy of the access packet directly to the requestor.

If the requestor can reconstitute the key, she can use it to access the service in question. In the case of data, the access key may be used to decrypt the actual data or unlock a lock box of other keys. In the case of services, the key can be used like a ticket to prove the requestor has been authorized to use the service. Many other possibilities exist because the protocol does not assume any specific form for key. It is merely a string of data.

Like Woo and Lam's proposed scheme [13], the Maille Authorization protocol relies on a separate authentication service to verify the identity of all requestors. One possible choice is the Maille Authentication system [7], which offers the advantage of offering similar resistance to inside attacks. Coupling our system with an authentication system with single points of failure, would greatly reduce the total trust warranted by the system. A simple identity theft would then allow any attacker to acquire authorization to any service.

The remainder of this paper is organized as follows. Section 2 reviews previous work related to distributed authorization. Section 3 introduces the Maille Authorization protocol. Section 4 presents an analysis of the protocol. Section 5 presents experimental findings from simulation. Section 6 has our conclusions. Finally, Section 7 sketches several directions for future work.

2. Related Work

Authentication in both centralized and distributed computing systems has received much attention over the last decade [1, 4, 6, 10, 14, 15]. In contrast, authorization has received little attention in a distributed context aside from defining policies [2, 3, 5, 12].

Unfortunately, very little work has been done to define frameworks for handling authorization information in a distributed system, or even centralized systems. Most systems implement local ad-hoc authorization schemes that do not rely on global frameworks. Even large distributed systems such as Globus rely on ad-hoc local management of authorization policy information [6].

Woo and Lam [13] proposed a framework for authorization in distributed systems. Their system relies on a fixed set of authorization servers. A service chooses one of the authorization servers to handle its authorization information. A more recent approach [11] uses PKI and specialized certificates but still remains centralized.

3. Authorization Protocols

Most authorization systems rely on the service host to act as a trusted gatekeeper. This requires the service owner to trust the host. It also results in a fundamental weakness. If the host is compromised, access can be granted to unauthorized users. Further, if replication is used, the chance of at least one host being compromised increases. The chance of the service being unavailable is reduced, while the chance that an attacker can be authorized increases.

To address these problems, our Maille Authorization protocol is designed to withstand multiple Byzantine failures such that a quorum is required before a user can obtain access to a service. The result is a protocol that tolerates many normal failures, and multiple inside attackers.

The Maille Authorization protocol suite is made up of several simple protocols carried out between nodes with no global knowledge. The following notation is used throughout the remainder of this paper:

- A, B, C represent specific nodes in the network.
- X and Y represent arbitrary nodes in the network.
- *f*(...) is a message, containing, among other, the parameters specified between the parentheses.
- **P** represents a service.
- **x** represents a specific action on a service.
- P_x represents an access key to perform action **x** on service **P**.
- P_{xi} represents the *i*th part of key P_x if decomposed with *t* of *n* threshold cryptography.

The Maille protocol operates in a peer-to-peer network. Each node in the network has a set of trusted peers with which it has a preexisting trust relationship. How these relationships are formed is beyond the scope of this paper. The result of two nodes A and B being peers is that A and B have exchanged public keys and can pass messages securely.

Within the Maille Authorization protocol, a service is uniquely identified by the pair (owner, service name) or (X,**P**). Each service has a non-empty set of actions: $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$. A service owner, a requestor's name and a specific action uniquely identify a key: P_x.

If the protocol states that a secure channel must be established, it is assumed that data flowing over that channel cannot be overhead by a third party or be altered without detection by the receiver.

The following message is defined within the protocol: *ar*(A,**P**,**x**,*ar_id*) - an access request by node A for the access key P_x. The *ar_id* field is a random string selected by the requestor, unique to each *ar*(…) message.

Each node in the peer-to-peer network maintains the following data structures: *ar_cache* – a list of all *ar*(...) messages seen recently and *access_lists* – a list of all access packets the node in question has available to answer *ar*(…) messages.

All entries in the *ar_cache* can be retrieved by the tuple (*requestor*, *target service*, *action*, *ar_id*). Would the random string not be part of the cache key, rogue nodes could easily flood the network with false *ar*(…) messages for a node they wish to deny service thus causing other nodes to drop valid *ar*(…) messages.

3.1. The Maille protocols

The Maille Authorization suite is made up of four protocols. Service owners use the *key distribution protocol* to scatter access packets onto the network. Nodes requiring authorization use the *authorization request protocol* to acquire the needed access key. All nodes in the network carry out the *authorization request propagation protocol* to forward *ar*(…) messages in an efficient manner. Finally, nodes holding access packets use the *authorization verification protocol* to verify that the key component P_{xi} should be returned to the requestor when an *ar*(…) message is received.

A. Key Distribution Protocol

When some service owner wishes to protect service **P**, it does the following for each action **x** :

1. Create a list of entities authorized to perform action **x**.
2. Create a random, secret *ap_id*.
3. Create a key P_x for action **x** on service **P**.
4. Decompose P_x using *t*-of-*n* threshold cryptography to create key parts $P_{x,1}, \dots, P_{x,n}$.
5. Create *n* access packets from the *n* key parts, each containing the authorized list, the *ap_id*, the service owner, the action and one of the distinct *n* key parts, P_{xi}.
6. Select *n* nodes at random from the network.
7. Send each selected node one distinct access packet through a secure channel.

Nodes receiving an access packet from a service owner authenticate the sender to insure she is the listed owner of the service and, if successful, simply add it to their *access_list* replacing any access packet for action **x** on service **P** with the same owner that might already exist.

How the service owner selects the *n* random nodes from the network is not specified. She may use a cache of observed nodes, or some outside knowledge. For best security, the nodes should be selected at random from as large a set of candidates as possible to reduce the risk of rogue nodes receiving access packets. Also, the service owner may need to maintain the list of nodes holding access packets so she can later change or invalidate the access packets.

B. Authorization Request Protocol

When node A wishes to acquire the access key P_x to perform action **x** on service **P**, it does the following:

1. Node A creates an access request: *ar*(A,**P**,**x**,*ar_id*) and send it to all its peers.
2. Node A collects responses until a predefined timeout expires.

Once the timeout has expired, node A may reassemble some keys P_x, P'_x, P''_x… Because more than *t* key parts may have been received, more than one key may be represented either by accident or by an attacker's design.

Node A groups the key parts by their *ap_id* and owner. Any set of key parts that has less than *t* members is discarded. Node A then uses these sets of key parts to construct possible access keys P_x, P'_x, P''_x … Each possible key P_x, P'_x, P''_x … is then tried on service **P** according to **P**'s own protocol until a good key is found or all keys have been tried.

If none of these keys works and some sets of key parts had more than *t* members, that set is likely to contain corrupted key parts. The requestor will then try all possible combinations of *t* key parts from that set. If this process results in a new key, that key will be tried on **P**.

While this procedure might appear cumbersome, most sets of returned key parts will only contain parts from one key. Multiple keys will only occur when either some nodes have out-of-date access packets, or an attacker is attempting to deny service.

In the first case, the *ap_id* of the out of date key parts will differ from the current *ap_id* so those parts will be grouped separately and will likely not have *t* parts available to assemble a key. The second case is discussed below in the Byzantine failure subsection.

C. Authorization Request Propagation Protocol

When some node X receives an access request *ar*(A,**P**,**x**,*ar_id*) from one of its peers Y, it does the following:

1. Node X checks *ar_cache* to see if it contains *ar*(...) identified by the tuple (A,**P**,**x**,ar_id) If so, it stops.
2. Node X adds *ar*(...) to the *ar_cache*.
3. Node X checks to see if *access_lists* contains an entry for action **x** on service **P**. If so, it carries out the Authorization Verification Protocol (below) and stops.
4. Node X forwards *ar*(...) to all peers except Y.

D. Authorization Verification Protocol

When a node X receives an access request *ar*(A,**P**,**x**,*ar_id*) for which it has an access packet, it does the following:

1. Node X verifies that A is listed in the access packet as allowed to perform action **x** on service **P**. If not, it stops.
2. Node X performs an Authentication against A to verify its identity. If authentication fails, X stops. Otherwise, X establishes a secure channel with A.
3. Node X sends node A a copy of the access packet, minus the authorized list, directly over a secure channel.

4. Protocol Analysis

In general, the system relies on the fact that all messages flow over secure channels. In the case of *ar*(…)

message propagation, only the two peers involved in each propagation can read or write the data being sent. An *ar*(…) message will flow undisturbed from peer to peer unless some node inside the network corrupts it.

4.1. Key Structure

The Maille Authorization protocol assumes no specific key structure. This allows for arbitrary keys to be used to secure services. Service owners are free to implement whatever key scheme they wish. For example, a service **P** may implement its authorization check as a simple nonce challenge against users trying to perform action **x**. This verifies that the user successfully underwent the authorization protocol and was authorized.

However, this type of system is inherently weak. Once a single key has been compromised, it can be passed around among rogue nodes, forcing the service owner to invalidate the key internally and redistribute new access packets containing a new key. Hence, access keys should always have a finite lifetime.

Service owners may wish to use a scheme similar to the key system used in Plutus [8]. New keys are generated such that the old versions of the key can be recreated by anyone with the current version, but old versions provide no information about the current key. Service owners can then rotate the key on a regular basis to protect new data only, thus removing the overhead of reencrypting all the data.

Service replicas not receiving the revised keys would continue to operate with the old keys. Users accessing such replicas need only generate a previous version of the key. This allows loose updates of security information among widely distributed replicas of a service and would be well suited for a data only service such as a distributed cache.

4.2 Efficient Authorization Request Propagation

Because Maille does not assume an underlying network structure, it must take steps to insure that the propagation of authorization requests does not lead to exponentially growing numbers of messages or to loops in the message propagation. To prevent both, nodes will only forward *ar*(…) messages they have not forwarded before. Nodes rely on a finite *ar_cache* of recent *ar*(…) messages to determine which messages should be forwarded. If the cache size is selected appropriately, the number of times an *ar*(…) message is forwarded will never exceed the number of connections in the graph. If an *ar*(…) message that is still active is flushed from a node's *ar_cache*, it is possible some extra forwarding of that *ar*(…) may occur.

4.3. Byzantine Failures

Rogue nodes within the network can subvert the system in two ways. First, they can attempt to gain access to services for which they are not authorized. To do this, a rogue node C must convince *t* nodes holding access packets for action **x** on service **P** that it is authorized to perform action **x** on service **P** when it is not.

Node C may try to fool the authentication system so that it can simply request authorization as some node that is authorized (i.e., identity theft). This protocol assumes an authentication system that cannot be easily defeated.

Alternately, C may try to collect *t* key parts by eavesdropping on other valid authorization requests. Because nodes holding access packets pass key parts directly to the requestor via a secure channel, C must attack the underlying cryptography of that channel. This is assumed to be practically impossible.

Second, rogue nodes can attempt to prevent otherwise valid authorization requests from succeeding. To do this, a rogue node can simply not forward *ar*(…) messages. However, this is no different than natural failures. The redundant nature of the Maille protocol will limit the impact of this type of attack.

The rogue may also try to answer *ar*(…) messages with false access key parts. This will result in a pollution of the reassembled key. It is conceivable that some threshold cryptographic system could detect this type of failure. Maille includes some mechanisms to overcome this type of attack without relying on the underlying threshold cryptography. There are two scenarios to consider.

The first scenario is that the rogue does not know the correct *ap_id*. The random secret *ap_id* included in valid access packets is used to group key parts. In the event of multiple collaborating rogues, a requestor might even receive enough erroneous responses to assemble more than one key. However, because only those nodes that hold valid access packets know the true *ap_id*, one of the reassembled keys will be the valid key. The requestor need only try each key in succession, until the real key is found.

The second scenario occurs if a rogue is one of the nodes holding a valid access packet. It may intentionally return a bad key part P'xi but with the correct ap_id. Unless the underlying threshold cryptography can detect this pollution, authorization may initially fail. If this happens and the requestor has received more than t key parts, the requestor can attempt to determine which key parts are bad. Assuming k key parts were received, the requestor A may attempt every possible t of k combinations.

Because *t* will generally be small compared to computation power available, this does not pose a

TABLE 1 – Average percent of access packets returned for all combinations of parameters. Due to computational time requirements, the 20 peers, n = 40, simulation runs for networks with 100,000 could never be completed.

Percent Access Packets Returned

		Network Size			
Peers	n	1000	10000	100000	**Average Result**
5	5	0.996	1.000	0.980	**0.992**
	10	1.000	0.997	0.970	**0.989**
	20	0.999	0.998	0.983	**0.993**
	40	0.999	0.997	0.977	**0.991**
10	5	0.992	0.980	0.934	**0.969**
	10	0.996	0.980	0.944	**0.973**
	20	0.996	0.976	0.943	**0.971**
	40	0.995	0.977	0.937	**0.969**
20	5	0.928	0.910	0.906	**0.915**
	10	0.940	0.931	0.909	**0.927**
	20	0.946	0.924	0.896	**0.925**
	40	0.936	0.926		**0.931**

computational problem. All false key part attacks can be handled as in the second scenario. However, for efficiency, the secret *ap_id* mechanism is included.

A more robust method, where the service owner signs the access packets prior to distribution, is possible. Any requestor can easily and with a high degree of certainty verify that the access packet is valid and unaltered. This would incur several (depending on t and n) additional authentications. The additional overhead seems unwarranted give that the problem should be rare and can be dealt with locally by the requestor.

5. Simulation Results

To test the protocol, an event-based simulator was implemented. It performs the actual protocol without modeling the underlying hardware and software. To model the race conditions often found in real world distributed systems, each message propagation takes a small random amount of time.

In all experiments, the simulator creates a random network of a user-defined size. The network is created such that all nodes have m or m-1 peers. This m value is known as the *peering* of the network. When the network is created a single service/action pair is automatically created. The number of access packets for the distributed service is also a user-defined parameter n specifying the total number of key parts used by the threshold cryptography.

5.1. Experiment 1 – Normal Operation

The primary purpose of this experiment is to verify that the protocol functions as expected and to establish a baseline for experiment 2. We ran the simulator on all combinations of the parameters, that is, n = 5, 10, 20, 40, m = 5, 10, 20 and *netsize* = 1000, 10000, 100000. Each combination of parameters was tested on ten different networks, with ten independent authorization requests per network. The results of the experiment are summarized in Table 1. The protocol performed as expected. Nearly 100 percent of access packets were returned with low standard deviation.

5.2. Experiment 2 – Natural Failures

Ideally, a node sending a request in a network where s percent of the total number of access packets it requested. For example if 10 percent of the nodes have failed, one would expect that on average 10 percent of the access packets would be unavailable. Any deviation from this is likely to be caused by problems with *ar*(…) message propagation due to node failure and the resulting problem of finding paths from the requestor to the nodes holding access packets.

We used the same simulator as in experiment one to test the performance of the network with various failure rates. Only networks of 10,000 nodes were tested. The

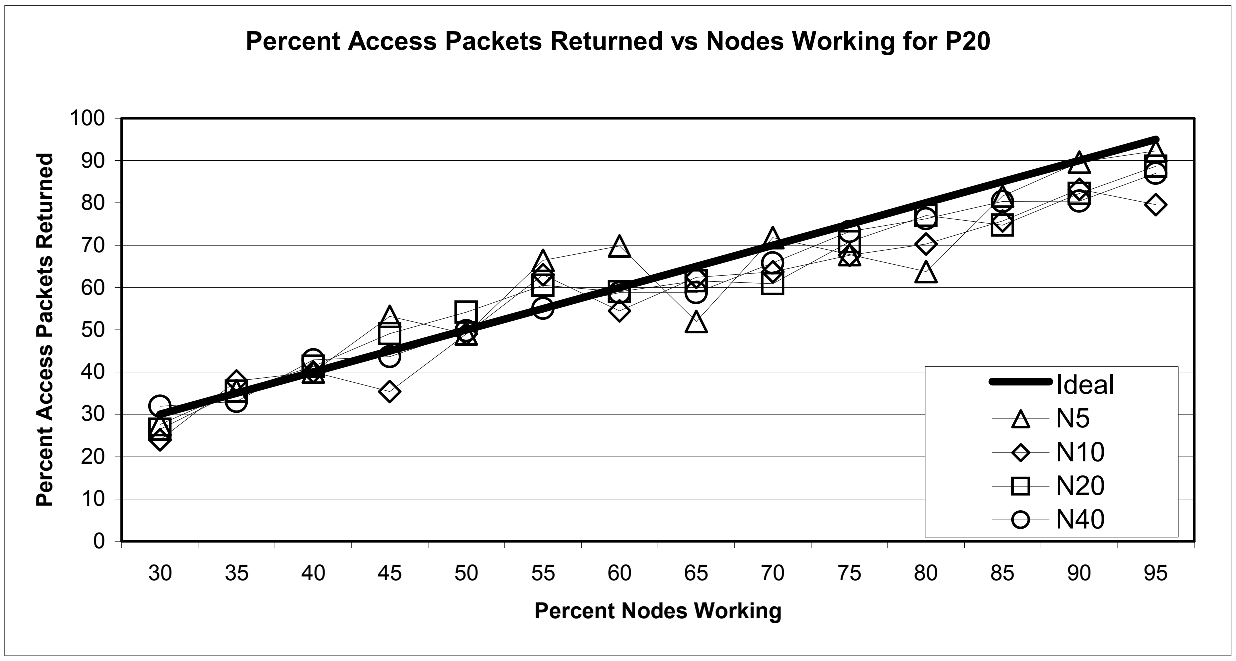

Figure 1. Percent of access packets successfully retrieved compared to percentage of nodes functioning for networks with a peering of 20. Results for 5, 10, 20 and 40 total access packets are shown.

parameters *n* and *m* were varied exactly as in experiment 1. A new parameter *f*—the percent of all nodes that have failed—was added. All values of f from 5 to 70 percent with 5 percent increments were run so that all possible combinations of the parameters n, m and f were tried. As in experiment 1, each time the simulator was run, 10 requests were simulated, and each parameter set was run 10 independent times for a total of 100 requests per parameter combination.

To analyze the results, we compared the percentage of nodes not failed to the actual percentage of access packets that were retrieved. Figure 1, 2 and 3 are summaries of the results.

Analysis of the data showed that the network's performance with failures is not greatly affected by the parameters tested. Networks with small peering (*m=5*) diverge from the ideal at high failure rates (over 50 percent). There was a slight trend for the standard deviation to increase as the failure rate increased.

The parameter *n* does not seem to affect the average return rate, but did show increased standard deviations at high failure rates with *n* equal 5. This leaves service owners to decide the amount of overhead they wish to incur creating and maintaining access packets in general. In networks with high failure rates, low values of *n* will lead to more erratic performance.

Overall, our data show that a service owner is free to choose any value of *n* without worrying about the impact of her choice on the protocol performance. They also indicate that the protocol is not sensitive to network size or peering over a wide range. The network requires little tending for the protocol to function.

6. Conclusions

The Maille Authorization protocol provides a way for widely distributed services to use authorization without requiring service owners to trusting each replica of the service with all authorization information. It distributes instead the authentication duties and data to a small subset of network nodes and relies on threshold cryptography to protect itself against multiple simultaneous Byzantine failures of the authentication nodes.

As a result, the Maille Authorization protocol is immune to the Byzantine failure of any single node. Further, because a qualified quorum of nodes holding the access packet for the service/action pair is required, no small number of Byzantine failures can improperly grant or deny authorization.

Traditional ad-hoc authorization systems required that the service owner trust the hosts of the replicas. In distributed systems this may not always be possible. For example, someone may wish to create a distributed database using spare hard drive space on personal computers. Clearly, the owner of the database cannot trust every personal computer to enforce all authorization policies. With the Maille protocol, the data need only be

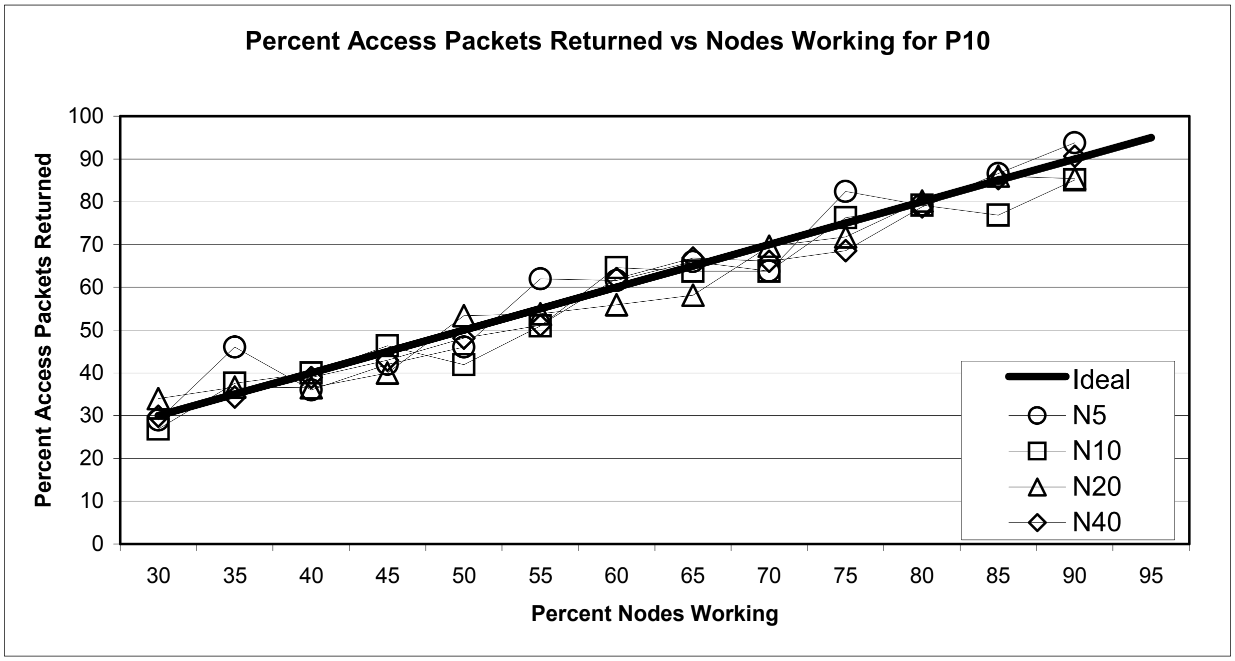

Figure 2. Percent of access packets successfully retrieved compared to percentage of nodes functioning for networks with a peering of 10. Results for 5, 10, 20 and 40 total access packets are shown.

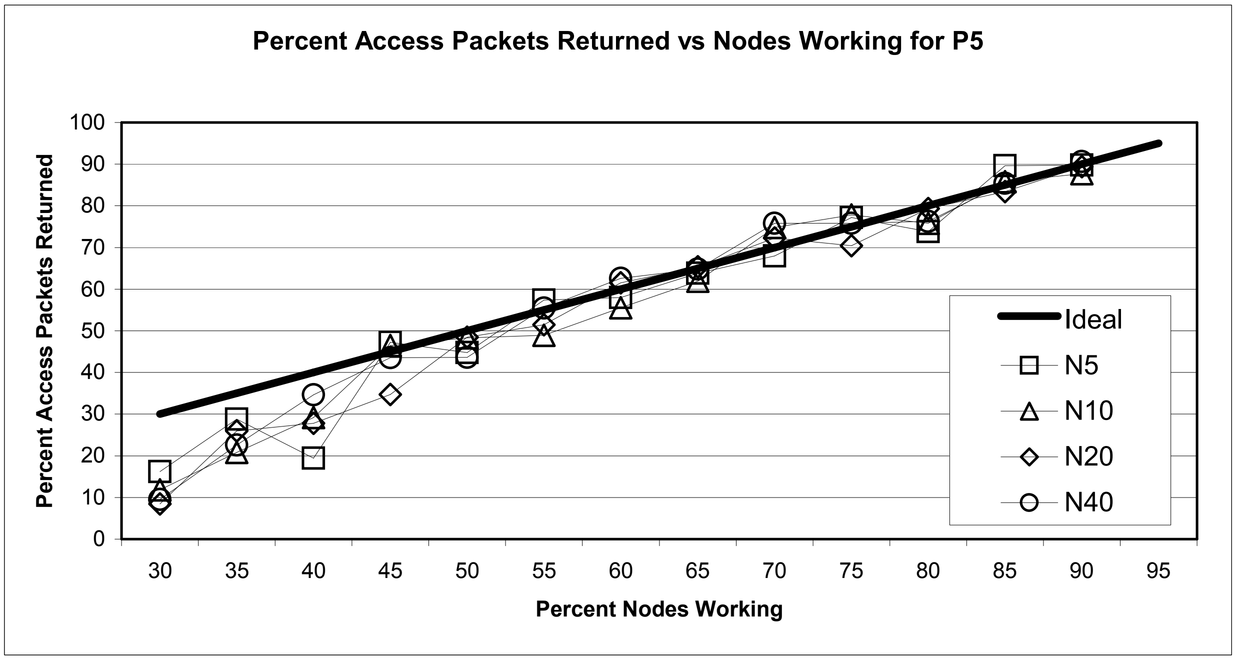

Figure 3. Percent of access packets successfully retrieved compared to percentage of nodes functioning for networks with a peering of 5. Results for 5, 10, 20 and 40 total access packets are shown.

encrypted such that the authorization process gives authorized users the required key. This decoupling of hosting from authorization is significant for pervasive computing and other distributed applications.

To date, most authorization research has focused on policy definition, and assumed that the service replicas themselves will perform all authorization tasks and retain the necessary information and policies to grant or deny access. Such systems only assume reliable authentication, but require the service owners to trust the hosts of their replicas. For widely distributed services, such as a wide area grid cache, the number of service replicas may be prohibitively large. Just keeping the authorization information on all replicas in sync may prove impossible.

7. Future Work

During the simulation, we realized that the stop at the end of step 3 of the authorization request propagation protocol might be impairing the network's ability to function with high failure rates. A better solution would be to always perform step 4 regardless of the presence of the requested access packet. Due to computational time constraints, the slightly altered version of the protocol was not tested.

Additionally, we would like to incorporate Maille Authorization with the Maille authentication protocol [7] and measure performance of the pair with more realistic simulations that include several simulated distributed services.

References

[1] Adams, C., and Lloyd, S., 1997, Profiles and Protocols for the Internet Public-Key Infrastructure, *Proc. 6th IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 220–224.

[2] Ahn, J., and Sandhu, R., 2000, Role-Based Authorization Constraints Specification, *ACM Transactions on Information and System Security*, **3**(4):207-226.

[3] Bertino, E., Bettini, C., and Samarati, P., 1994, A Temporal Authorization Model, *Proc. 2nd ACM Conf on Computer and Communications Security*, pp. 126-135.

[4] Bird, R., Gopan, I., Herzberg, A., Janson, Ph., Kutten, S., Molva, R., and Yung, M., 1995, The KryptoKnight family of light-weight protocols for authentication and key distribution, *ACM/IEEE Transactions on Networking*, **3**(1):31–41.

[5] Bonatti, P., Vimercati, S., and Samarati, P., 2002, An Algebra for Composing Access Control Policies, *ACM Transactions on Information and System Security*, **5**(1):1-35.

[6] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S., 1998, A Security Architecture for Computation Grids, *Proc. 5th ACM Conference on Computer and Communication Security*, pp. 83–92.

[7] Fritz, A., and Pâris, J.-F., 2004, Maille Authentication: A Novel Protocol for Distributed Authentication, *, Security and Protection in Information Processing Systems* (Y. Deswarte, F. Cuppens, S. Jajodia and L. Wand, eds.), pages 309–322, Kluwer Academic Publishers, 2004.

[8] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K., 2003, Plutus: Scalable Secure File Sharing on Untrusted Storage, *Proc. 2nd Conference on File and Storage Technologies*, pp 29-42.

[9] Shamir, A., 1979, How to Share a Secret, *Communications of the ACM*, pp 612-613.

[10] Steiner, J G., Neuman, C., and Schiller, J. I., 1988, Kerberos: An Authentication Service for Open Network Systems, *Proc. 1988 Winter Usenix Conference*, pp. 191–201.

[11] Thompson, M., Essiari, A. and Mudumbai, S., 2003, Certificate-Based Authorization Policy in a PKI Environment, *ACM Transactions on Information and System Security*, **6**(4):566-588.

[12] Varadharajan, V., and Allen, P., 1996, Joint Actions Based Authorization Schemes, *ACM SIGOPS Operating System Review*, **30**(3):32-45.

[13] Woo, T. and Lam, S., 1993, A Framework for Distributed Authorization, *1st ACM Conference on Computer and Comm. Security*, pp. 112-118.

[14] Zhou, L., Schneider, F. B., Van Renesse, R., 2002, COCA: A Secure Distributed Online Certification Authority, *ACM Transactions on Computer Systems*, **20**(4):329-368.

[15] Zimmermann, P., 1995, *The Official PGP User's Guide.* MIT Press, Cambridge, MA.