Towards threat-adaptive dynamic fragment replication in large scale distributed systems

Roberto Di Pietro[1], Luigi V. Mancini[2], and Alessandro Mei[2]

[1]Università di Roma Tre
Dipartimento di Matematica
l.go S. Leonardo Murialdo 1, 00146 - Roma, Italy
dipietro@mat.uniroma3.it

[2]Università di Roma "La Sapienza"
Dipartimento di Informatica
via Salaria 113, 00198 - Roma, Italy
{mei,mancini@di.uniroma1.it}

Abstract

In this paper, we consider new issues in building secure p2p file sharing systems. In particular, we define a powerful adversary model and consequently present the requirements to address when implementing a threat-adaptive secure file sharing system. We describe the main components of such a system: An early warning mechanism to perform pre-emptive actions against new vulnerabilities; a mechanism to sanitize corrupted nodes; a protocol to securely "migrate" data from non-safe nodes; and an efficient dynamic secret sharing mechanism.

1 Introduction

The Internet and the WEB have defined an inherently distributed and insecure environment for information sharing. New forms of content distribution use resources provided in a fully interconnected environment which is changing our life. Information sharing is getting more and more importance for many reasons, for private and public purposes, for research progression and for business. Current technology should enable its fruition.

Peer-to-peer (P2P) protocols allow to build networks of huge size, with thousands or millions of cooperating users. P2P networks can be scalable, resilient and efficient, usually without the requirement of centralized servers, in a completely distributed fashion. P2P protocols provide underlayers for a large variety of services, like network storage, content distribution, web caching, searching, indexing and more. Among these services, a storage utility like PAST [24] is attractive for several reasons. First, it can exploit the diffusion and heterogeneity (in geography, ownership, administration, jurisdiction, etc.) of the nodes in the Internet to achieve strong persistence and high availability. This obviates the need for physical transport of storage media to protect backup and archival data; likewise, it implies an enhanced degree of availability and throughput for shared data. A global storage utility also facilitates the sharing of storage and bandwidth, thus permitting a group of nodes to jointly store or publish content that would exceed the capacity or bandwidth of any individual.

In this scenario, P2P systems are natural candidates for the information sharing revolution, and are acquiring much importance and interest. In particular, a P2P system allows a (potentially) huge number of users to access different resources, distributed over the nodes of the system, in a uniform way regardless of the actual allocation. The main benefits are performance, scalability, high availability, fault-tolerance and reliability. However, security of the shared information is an issue. Many companies and organizations spend billions of dollars in order to guarantee security of their data and to fix information leaks caused by intruders or insiders's attacks. Information sharing is indissolubly tied with the necessity of securing the information itself.

The goal of our work is to build a system for information sharing that appear as a centralized system but is actually physically distributed among a large set of untrusted machines, according to the P2P paradigm. The shared information (e. g. a file) can be accessed by many individuals from many different locations that may be geographically distributed world-wide. Data can be accessed in two different modes, for reading and writing, and can be spread all over the network with an appropriate degree of redundancy. The system should provide security—in this paper we focus

This work was partially funded by the *WEB-MINDS* project supported by the Italian MIUR under the FIRB program.

on confidentiality and availability—without requiring physical protection, continuous administration, and burdensome tasks of centralized server environments. Indeed, in a P2P system every machine belonging to the system, henceforth *node*, performs some client-side activities (it interacts with the user and requests the access to the objects) and has some server-side responsibilities (file storage and access request service). It provides the location-transparent access typical of client-server architecture, but has the benefits of typical desktops and workstations, i. e. no need for centralized administration and low cost.

In this paper we sketch the architecture and mechanisms that can lead to the design of a threat-adaptive file sharing system. In particular, we devise a system that is robust against *active* and *silent* adversaries. The former type of adversary is able to compromise one or more nodes and collect all of their content (the object itself, any secret key, etc...). The latter is protocol complaint till it has compromised enough nodes to perform a successful attack. Stemming from this model of adversary, we describe the architecture components that are required to preserve the security feature of the system, that we assume to be confidentiality and availability. In particular, we sketch the need for the following mechanisms: Early warning, sanitization, secure migration, and secret sharing. These mechanisms will be described in detail in the following.

The remaining of this paper is structured as follows: Next section reviews a secure and efficient P2P file sharing system. This system will be referenced throughout the paper; Section 3 introduces the adversary model that threatens the security and confidentiality of the file sharing system; Section 4 introduces the required architecture components. In particular, in Subsection 4.1 the Alert & Sanitization mechanisms are introduced; Subsection 4.2 justifies the need for a migration mechanism of the shares hosted on class of nodes, while Subsection 4.3 describes the requirements for a secret sharing scheme. Section 5 reports on the related work in the literature. Finally, Section 6 presents some concluding remarks and indicates a few research directions.

2 Secure and efficient P2P file sharing system

An *active attack* is an attack launched by an adversary that is able to break into one or more nodes of the p2p system, potentially compromising any stored data. In order to break into a node, the adversary exploits a system weakness. Of course, there can be several kinds of such weaknesses and security holes: Poor system administration; bad users' practices like, for example, choosing easily guessable passwords; malicious insiders, and so on. Moreover, there can be different kinds of software and hardware flaws that can be exploited, for example with buffer overflow attacks. A possible solution to the problem of keeping data confidential is to use cryptographic techniques and encrypt the storage assuming that only authorized users are aware of the correct decryption key. Unfortunately, practice says that no encryption system is perfectly safe, so we need to assume that breaking a cryptographic system is not impossible. For example, the adversary could perform a dictionary attack or use mathematical crypto-analysis; or it could use social engineering techniques for retrieving an authorized user's secret key, whenever the user is not scrupulous enough in securing it. More importantly, encrypted storage rises not-trivial issues about *key management*. In fact, key management has always been a challenging problem for data sharing in large systems: encryption keys for storage have longer lifetime compared to keys for communication; they need to be securely stored and accessed only by authorized users, which can belong to different administrative domains. This raises the questions of how authorized users should obtain the decryption keys, who should be responsible for providing those keys, and to do it efficiently and securely. Further, encrypted storage calls for some *recovery mechanisms* and key escrow procedures, for the event of secret key loss or corruption. This mechanism could be, itself, a possible security hole.

In order to cope with active attacks without facing the above issues, [30] proposes the use of *data fragmentation* rather than encrypted data storage, and to store different fragments of the same object on different nodes of the system. There are many fragmentation mechanisms, like Rabin's information dispersal [21] and Shamir's secret sharing [26]. By using these mechanisms, it is possible to fragment a file into n pieces, called *shares*, such that any m of them are enough to reconstruct the file, while $m-1$ shares give no information (or give no information to every computationally bounded adversary). The idea is to partition the nodes into t "classes", $t \geq n$; nodes from different classes should be heterogeneous in such way that any active attack that is successful with nodes in a class, should be ineffective with nodes in other classes. A similar, but stronger, assumption has been done in [28], for example, where the system is completely heterogeneous—every node in the system is unique in software, administration, and so on.

The distributed system proposed in [30] is a collection of nodes organized in a tree (the physical tree). An example of such a system is shown in Figure 1. Clients connect to one of the nodes to read and write a shared file. When the client reads the file, it has to retrieve a copy of each of the t fragments. Hence, it must find at least one node from every class. When the client writes the file, it has to update every copy of every share. A key point to address in the system is how to partition the nodes into classes in such a way that it is possible to get high performance. Every client should be able to find a copy of every fragment relatively quickly. The

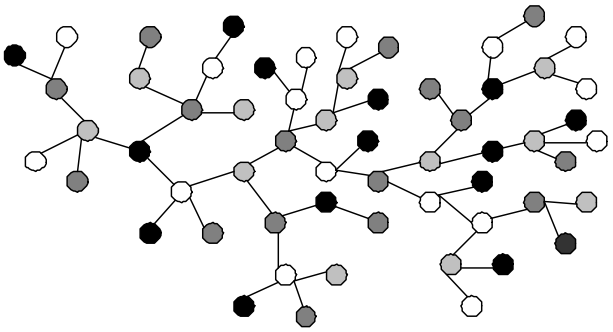

Figure 1. Partition of a network into 4 classes.

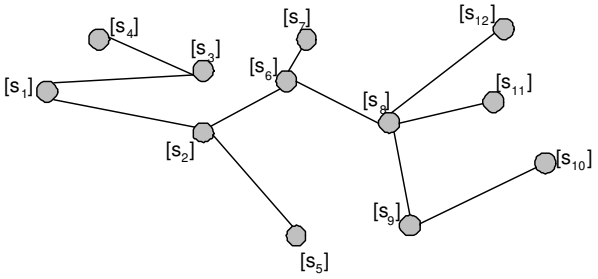

Figure 2. A logical tree.

problem of partitioning is solved by choosing a hash function $H : \mathbb{N} \mapsto \{1, \ldots, t\}$ and setting the class of node i to $H(i)$ (this means that the architecture of the node depends on its ID). With high probability, every client will be close to a node from each of the t classes. An example of this random partitioning is shown in Figure 1. The set of nodes in every class self-organizes in a distributed fashion into a logical network, that turns out to be a tree where, under proper hypothesis, logical edges (edges from nodes of the same class) can be simulated by using a path of at most $O(\log t)$ physical edges. So, performance of communications on the logical trees are provably efficient. Figure 2 shows one of the logical trees from the partition of Figure 1—the one composed by the light gray nodes. Finally, every share is assigned one of the classes in the distributed system, and consequently a logical tree. Then, a secure replication and migration protocol is run on each logical tree to replicate and move each share towards the optimal location given a fixed read/write activity on the file by the clients of the system. When $m = n = t$ and when the client activity is stable, [30] shows that the protocol converges to the optimal share allocation. For the details, the reader is referred to the original paper.

3 Adversary model

In our view, the *adversary* is a malicious individual that can misbehave in many flavors in order to compromise both the availability and the confidentiality of the file sharing system. Specifically, the adversary we define has the following goals: a) to compromise the integrity of the shares hosted on a specific server; b) to collect a set of shares such that it can recover the file. Further, more malicious individuals can collude, by combining their resources, in order to actively attack the service. In the following, we follow a conservative approach and consider that all the adversaries collude; we denote such a group of conspirators as a single adversary. An *active attack* is an attack launched by an adversary that is able to break into one or more classes of nodes, potentially compromising those nodes.

Threats cannot be realistically avoided and, in general, it is not possible to foresee the vulnerabilities that could be exploited by the adversary in the future, as well as the extension of the attack (that is, just a node, all the nodes for a given class, several nodes spanning several classes). For this reason, we allow the adversary to be able to corrupt up to a given number of nodes.

Once a server is corrupted, confidentiality and integrity of any private and public information stored on the server are assumed to be compromised, and these data can be recorded or leaked to third parties, as well as arbitrarily modified. Moreover, the adversary can re-program the server, changing the protocol it was supposed to follow, making the node unavailable, and so on. In [15, 14], two adversary models are presented, to characterize the *power* of these adversaries:

> **Long-term constrained adversary**: the adversary can break into and control at most $t - 1$ nodes, during the *entire life* of the network;
>
> **Short-term constrained adversary**: assuming that the time is divided into intervals, the adversary can break into and control at most $t - 1$ nodes during any *time interval*.

Note that, even though victims can be arbitrarily chosen, in both of the models above t is a fixed parameter, expressing the system's robustness against active attacks. In the long-term constrained model, the adversary has the entire life of the network to attempt to disrupt or deny the service. Gradual breaks into subsets of nodes over a long time period are feasible. In the short-term constrained model the adversary has just an arbitrary short time-window to compromise enough nodes, that is the adversary is more constrained.

Proactive secret sharing [10] is a secret sharing scheme that periodically renews shares, without changing the secret (the file), such that any information learned by the

adversary becomes obsolete after the share-renewal protocol completes. The scheme assumes that participants are not malicious, that is they destroy their previous shares as soon as they get the new ones. By periodically refreshing the single shares, the time window in which the adversary needs to corrupt the nodes is supposed to decrease dramatically, making the network more robust. The assumption made is that it is possible to recover a compromised node via some administration technique (for example by resetting hardware and system configurations, reloading the code and so on), thus removing the adversary from corrupted nodes. This is done either when compromission is detected (whenever detection is possible) or at the beginning of every interval the time is divided into, before the share-renewal phase. Note that this kind of node recovery could be classified as *weak*, since just reloading the proper code on a node does not eliminate the vulnerability the adversary used to break into the node; hence, the adversary could immediately regain control of the node. We would like to take into account a form of *strong* recovery as well. This recovery is performed when the vulnerability that was exploited by the adversary is eliminated (for instance, installing a patch to the OS, or a fix to the application's code). We consider *strong* recovery a realistic assumption, since patches and fixes are nowadays delivered, or at least signalled, in automatic ways. Hence, we can assume that if we can apply a *strong* recovery on a node (we say that the node is *sanitized*), the same sanitization can be extended to all the nodes in the network that belong to the class of the sanitized one, and eventually the adversary is expelled from a certain class of servers. Sanitization mitigates one of the major issue in these scenarios: the possibility for an adversary to break in the network and act *silently*. Nodes under control of the adversary could exhibit a correct behavior, as long as their number is not enough to harm the service. A class that is just sanitized is safe; however, a sanitized class could be compromised again if a new vulnerability for that class is found by the adversary.

This is likely to occur for large-scale long-lived networks, because the probability to compromise enough classes increases with the number of nodes. According to [10, 14], thanks to pro-activity, this scenario is modeled by a short-term constrained adversary, but the adversary, actually, still has the entire life of the network for compromising enough nodes.

One possibility to cope with this kind of a attack is proposed in [29] within the context of distributed signatures. This proposal heavy relies on the algebraic properties of RSA, and its general applicability is still a concern.

In this paper, to thwart the proposed adversary model, and in particular the *silent* adversary, we propose to dynamically increase the *confidentiality* of the service, that is the number of shares needed for reconstructing of the file. Moreover, since *availability* is a plus for distributed file systems, we also increase the number of redundant shares.

4 Architecture components

In this section we describe the components of the architecture that will provide our system both confidentiality and availability. Further, we highlight a few directions to implement these mechanisms.

4.1 Alert & Sanitization components

The most serious threat in our model is given by the assumption that the adversary is *silent*, as we have discussed in Section 3. Indeed, if the adversary continue corrupting nodes belonging to different classes, and the number of classes does not increase, at a certain point the adversary will be able to gain access to the file.

To cope with this adversary, our architecture needs three mechanisms:

- *early warning*: This mechanism reports which classes can be compromised, that is the classes for which a vulnerability has been identified;
- *sanitization*: A way to sanitize the possibly compromised systems. We take a conservative approach and manifest our interest for sanitizing systems for which it is possible *strong* recovery only. Indeed, sanitization that would lead to a *weak* recovery does not provide any assurance on the fact that the adversary could not take control of this class of nodes;
- *secure migration*: In the time frame between the early warning for a class of nodes and its sanitization, the security of the file is decreased. To keep the same security assurance, it is possibile to "migrate" the shares of the file towards safe classes by forcing a proper rewrite of the file.

Note that to-date there is an increasing interest of the security community to span and implement early warning systems. For instance, the information provided by existing CERTs [1] as well as public and private Institutes [2] provides an effective way to implement early warning and sanitization.

Note that once a node has been informed that its class could be compromised, it must be decided whether, how, and where to migrate the shares. Indeed, the fact that its class *could* be compromised does not necessarily imply that it has been compromised. Migration could effectively preserve the availability and confidentiality degree of the system. Details about the migration existing strategies will be discussed in next subsection.

However, if migration intuitively helps in preserving the confidentiality of the scheme, an active measure has to be adopted to increase the number of classes that can be considered *safe*, where we define *safe* a class that is not subject to a known vulnerability. We can assume that once a certain class has been demoted to *non-safe* due to the discovery of a vulnerability, the same class could be promoted to *safe* if the vulnerability is eliminated.

4.2 Migration mechanism

Assume that a certain class has just moved from a *safe* to a *non-safe* state. What are the actions that should be undertaken to preserve *confidentiality* and *availability*? Indeed, usually a vulnerability is reported after a system with the same weak component has been attacked (this system has been subject to a so called *zero-day* attack). If our distributed system has been the targed of a zero-day attack, it is not enough to migrate all the shares from the target class to a safe one. Indeed, a reasonable and conservative assumption is that all the information stored in any of the nodes of the compromised class (not only the first target of the attack) is lost. So, migration does not help. In this case, the system should issue a forced rewrite of the file, taking care that noone of the new n shares is stored in a non-safe class. Depending on t, n, and m, and on the number of safe classes in the system, this is not always possible. Note that the new shares are completely independent of the previous, possibly compromised, shares of the same file. In this way, the confidentiality and availability of the file is preserved.

When the system is not the target of a zero-day attack, instead of performing a rewrite of the file, that is an expensive operation, it is possible to consider a simple migration of the only share (and all of its copies) from the class that just turned to non-safe towards a class that is classified safe, that is, for which no vulnerabilities are known. This way, if the system has not been compromised yet, we increase the chances of preserving both *availability* and *confidentiality*. This operation is much faster and less expensive, though does not garantee the same degree of security.

4.3 Secret sharing

In the previous section we have seen that, in order to design a file sharing system that is adaptive to the threat from a dynamically evolving adversary, it is required a mechanism to migrate the share. However, note that only migrating the shares would provide no assurance that the file system degree of security is preserved. Indeed, when migrating a share, we assume that the class is in a *non-safe* state, that is, it is subject to a vulnerability, but not necessarily it is has been already compromised. The shares are migrated to the nodes belonging to one or more classes that are in a *safe* state.

A secret sharing mechanism should help in thwarting this depletion of security. We have seen in Section 2 that the files to be shared are split into shares according to some (n, m) mechanism, that is at least m out of n shares are required to recover the original file, like in [26]. The secret sharing that should be adopted should allow to increase the either the value n or both values m and n — for instance, to $(m, n+1)$ or $(m+1, n+1)$. Note that the relationship $t \geq m \geq n$ should be kept. In this way, the number of classes increases and this should balance the gain achieved by the adversary when corrupting a class.

4.4 Caveats

In this subsection we point out a few caveats for the migration and secret sharing mechanisms.

Migration mechanism

What is left in the migration mechanism above specified, is to decide how migration should be addressed. One possibility would be to migrate the shares towards a single class of safe nodes; however, note that this solution would not necessarily preserve the security properties. Indeed, shares could migrate towards a class that is subject to a zero day attack. Further, a few performance related issues arise: Destination nodes could be overloaded—above the average storage could be required, as well as above the average queries could be addressed to these nodes. To limit the extent of the possible compromise and to provide a balanced allocation of shares, it follows that the shares should migrate towards the class of server that are still in a *safe* state.

At this point, we can assume that once a class has been marked as *non-safe*, all the shares are migrated. This triggers an interesting question: What happens when sanitization intervenes? That is, once a class of nodes is sanitaized, all the nodes in this class are promoted to a *safe* state. Hence, this class contributes to increase the degree of service availability; that is this class can now start receiving shares from other servers (to receive shares that migrates from nodes that are demoted to a non-safe state, or to implement load balancing, if needed). A few points would need better understanding: Which shares can be assigned to this renewed class of servers? One possibility could be to migrate to these nodes the same shares that migrated away when these nodes where demoted; but if this is the solution, what if the same shares have been migrated twice, since the destination class has been marked *non-safe* as well? This issue can be tackled in two ways: Via an efficient way to track share distribution, or assigning shares to nodes independently from their generation history, for instance in a pseudo-random way like in [29].

Secret sharing
The features a secret sharing mechanism should provide can be identified as follows:

- *dynamic:* The secret sharing mechanism should allow the threshold to increase, but also to decrease. While in Section 4.3 we have explained why the threshold should increase, note that increasing the number of shares the file has been split into would require writing operations to be more expensive. Hence, if the security conditions of the network indicate a demoted adversary capability (this can be due, for instance, to an increased capacity in nodes sanitization), the overall number of split can be reduced, improving the performances of both reading and writing operations.

- *efficient:* Note that to read the file, a legitimate user has to access at least a share from m classes. The more the classes, the more the nodes that a user has to contact. Furthermore, the more the classes, the higher the overhead to regenerate the file due to a file-write operation. The latter consideration implies that the number of shares should be as small as possible in order to reduce the overhead associated to file operations. Furthermore, since writings require to re-generate all the shares, it is necessary that the share generation procedure be efficient.

- *verifiable:* This last property derives from a recent attack shown in the literature, the *pollution attack* [11]. The following example explains this attack: The adversary can take over a node without raising any alarm; then, to disrupt the service, it provides a corrupted share upon request. The requester, once it has collected the shares, tries to rebuild the file, but the attempt fails since the corrupted share does not enable the correct recovery of the file. The goal of *verifiable secret sharing* is to implement efficient solutions like the ones in [11, 8], that are suitable to a multicast environment, but that still need to be adapted to the completely distributed paradigm. A novel solution like the one in [29], could be applicable, but it is still questionable if the proposed method, that relies on the properties of the modular algebra inherited by RSA, could be efficiently generalized to support different cryptographic techniques.

5 Related work

Distributed file systems has been a long lasting research area. However, these systems generally focus on availability and often set out weak security (whenever taken into account) or low performance, or both.

The most famous distributed file system is NFS, whose specification has been made public by SUN in [17]. Basically, NFS passes most of the data in the clear, relying on insecure networks and trusted hosts. AFS [18] has been developed at the Carnegie Mellon University with the support of IBM. Security is provided by Kerberos [20] between clients and servers; files are stored in the clear. The CODA system [25] is the evolution of AFS. The main difference with its parent is the automatic replication system implemented in CODA, not provided by AFS, that improves on the availability of the system at the expense of consistency. CODA introduces specialized conflict resolution procedures. Another important feature is that it provides the possibility of disconnected operations. SPRITE [19] is not just a distributed system but a real Network Operating System, developed at Berkeley. Its file system, SPRITE LFS [23], uses replication and caching, and guarantees consistency among replicas, at the expense of performance decay.

xFS [5] decentralizes the file system service among a set of trusted workstation. The same approach is used in Frangipani [27], that employs a distributed RAID [12] rather than full replication. Frangipani has two layers, a distributed storage service called Petal [13], that optionally replicates data, and a set of symmetric servers. Both of the systems are server-less, i. e. the responsibility for files is spread over all the nodes. xFS, unlike Frangipani, has a predesignated manager for each file. In particular, Frangipani is meant to be used in a cluster of servers that share a common administration and can communicate securely (and trust each other).

Some security issues have been considered in several projects. SCARED [22] uses cryptography to authenticate users and end-to-end encryption of data moving through the network, without encrypting the data while stored. In [9] the system encrypts the storage at disk volume level and employs an emergency key recovery mechanism that uses Shamir's secret sharing [26] to distribute key shares to trusted individuals. Many systems provide authentication but not data confidentiality, so the user has to implement its own mechanism if willing to protect its files. Of course, in such systems it is not easy to provide file sharing. SUNDR [16] guarantees some important security properties even in presence of a compromised server, but it assumes that all the clients are trusted. Most of the systems addressing scalability are generally used for archiving data, rather than to provide collaborative environments in which information is read and written interactively. In this category we find large p2p file sharing systems like Napster [4] and Gnutella [3], that are suitable for finding files but do not explicitly replicate files nor determine storage location; and like Freenet [7], that replicates files near the points of usage; some storage systems employ the same philosophy, CFS [6] and PAST [24]. The typical limit of these systems

is that they assume read-only or one-publisher data, hence only the publisher of the file can modify it, while other users can only read it. CFS guarantees robustness, efficiency and load-balancing; availability is assured by replication without encryption, with the idea that files can be end-to-end encrypted by clients before storing; however, CFS does not provide any facility for information-sharing. PASIS [28] offers more features: It makes use of secret sharing and information dispersal [21] to provide confidentiality, availability and integrity, at the cost of performance. In PAST, replicas are placed on a diverse set of network nodes by a fault-tolerant and self-organizing routing and location infrastructure; the set of nodes storing replicas is determined pseudo-randomly.

6 Concluding remarks

In this paper we have set out the requirements, the issues, and the mechanisms that need to be addressed when implementing a dynamically secure file sharing system. In particular, we have described the main components of such a system.

This paper also paves the way to foster a few research issues: The sanitization mechanism would need a thorough analysis of its fundamental properties; indeed, the rate at which systems get infected and are sanitized would impact on the possibilities for the reliability of the system to improve or to lead the system itself to compromise. Another research issue steams from the early warning mechanism: Should it follow a push or a pull paradigm, and how to efficiently integrate this logical point of centralization into a distributed architecture? We are currently addressing the fundamental questions related to the sanitization mechanism.

References

[1] http://www.cert.org/.

[2] http://www.sans.org/top20/.

[3] The Gnutella website. http://gnutella.com.

[4] The Napster website. http://www.napster.com.

[5] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. *ACM Trans. Comput. Syst.*, 14(1):41–79, 1996.

[6] M. Blaze. A cryptographic file system for unix. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 9–16, New York, NY, USA, 1993. ACM Press.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[8] R. Di Pietro, S. Chessa, and P. Maestrini. Computationally, memory and bandwidth efficient distillation codes to mitigate dos in multicast. In *Proceedings of the* 1^{st} *IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, pages 13–22. IEEE Press, 2005.

[9] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst.*, 20(1):1–24, 2002.

[10] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or how to cope with perpetual leakage. In D. Coppersmith, editor, *Advances in Cryptology: CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 1995.

[11] C. Karlof, N. Sastry, Y. Li, A. Perrig, and J. Tygar. Distillation codes and applications to DoS resistant multicast authentication. In *Proceedings of the* 11^{th} *Annual Network and Distributed Systems Security Symposium (NDSS 2004)*, pages 37–56, February 2004.

[12] E. K. Lee, P. M. Chen, J. H. Hartman, A. L. C. Drapeau, E. L. Miller, R. H. Katz, G. A. Gibson, and D. A. Patterson. Raid-ii: A scalable storage architecture for high-bandwidth network. Technical report, University of California at Berkeley, 1992.

[13] E. K. Lee and C. A. Thekkath. Petal: distributed virtual disks. In *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 84–92, New York, NY, USA, 1996. ACM Press.

[14] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang. URSA: Ubiquitous and Robust Access Control for Mobile Ad-hoc Networks. *IEEE/ACM Transactions on Networking (ToN)*, 12(6):1049–1063, 2004.

[15] H. Luo and S. Lu. Ubiquitous and robust authentication services for ad hoc wireless networks. Technical Report TR-200030, UCLA, Dept. of Computer Science, 2000.

[16] D. Mazières and D. Shasha. Building secure file systems out of byzantine storage. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 108–117, New York, NY, USA, 2002. ACM Press.

[17] S. microsystems. NFS: Network file system version 3 protocol specification. Technical report, SUN microsystems, 1994.

[18] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith. Andrew: a distributed personal computing environment. *Commun. ACM*, 29(3):184–201, 1986.

[19] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, 1988.

[20] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):32–38, 1994.

[21] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[22] B. C. Reed, E. G. Chron, R. C. Burns, and D. D. E. Long. Authenticating network-attached storage. *IEEE Micro*, 20(1):49–57, 2000.

[23] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[24] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM Press.

[25] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

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

[27] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: a scalable distributed file system. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 224–237, New York, NY, USA, 1997. ACM Press.

[28] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliccote, and P. K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, 2000.

[29] G. Zanin, R. Di Pietro, and L. V. Mancini. Robust RSA distributed signatures for large-scale long-lived ad hoc networks. *Journal of Computer Security*, to appear.

[30] G. Zanin, A. Mei, and L. V. Mancini. A secure and efficient large scale distributed system for object sharing. In *Proceedings of the 23th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.