Survivable Information Storage Systems

The PASIS architecture flexibly and efficiently combines proven technologies for constructing information storage systems whose availability, confidentiality, and integrity policies can survive component failures and malicious attacks.

Jay J. Wylie
Michael W. Bigrigg
John D. Strunk
Gregory R. Ganger
Han Kılıççöte
Pradeep K. Khosla
Carnegie Mellon University

As society increasingly relies on digitally stored and accessed information, supporting the availability, integrity, and confidentiality of this information is crucial. We need systems in which users can securely store critical information, ensuring that it persists, is continuously accessible, cannot be destroyed, and is kept confidential. A *survivable storage system* would provide these guarantees over time and despite malicious compromises of storage node subsets. The PASIS architecture combines decentralized storage system technologies, data redundancy and encoding, and dynamic self-maintenance to create survivable information storage.

TECHNOLOGIES FOR SURVIVABLE STORAGE

Survivable systems operate from the fundamental design thesis that no individual service, node, or user can be fully trusted: Having some compromised entities must be viewed as the common case rather than the exception. Survivable systems, then, must replicate and distribute data and services across many nodes.

Survivable information storage systems entrust data persistence to sets of nodes rather than to individual nodes. Individual storage nodes must not be able to expose information to anyone; otherwise, compromising a single storage node would let an attacker bypass access-control policies.

To achieve survivability, storage systems must be decentralized and must spread information among independent storage nodes. To be manageable, survivable systems must monitor and repair themselves to avoid unchecked degradation over time.

Decentralized storage systems

Most existing distributed systems store information on, at most, a few servers, making them central so that they are targeted failure points for accidents and malicious attacks. Replacing centralized approaches with highly decentralized systems is a first step toward survivable information storage. Fortunately, much work has been done in cluster storage systems—such as xFS,[1] NASD,[2] and Petal.[3] These systems all provide applications with a single, unified view of the storage system while distributing or replicating data and metadata across many storage nodes. With a single-system image of storage, client applications don't need to deal with the complications of data distribution and service decentralization.

Decentralized storage systems partition information among nodes using data distribution and redundancy schemes commonly associated with disk array systems such as RAID (redundant array of independent disks)[4] ensuring scalable performance and fault tolerance. For example, the Petal system stores portions of a file on different storage nodes, replicating each data block on two nodes. When users need to access data, they contact a subset of the storage nodes in the system for the desired information blocks. Although decentralized storage systems were devised for scalability and tolerance of infrequent faults, their elimination of single failure points provides a starting point for developing survivable storage systems.

Data redundancy and encoding

Availability and confidentiality of information are primary goals of many information systems. Most systems enhance availability by providing full replication,

0018-9162/00/$10.00 © 2000 IEEE

IPR2023-00969
International Business Machines Corporation v. Digital Doors, Inc.
IBM's Exhibit 1026
Ex. 1026, Page 1

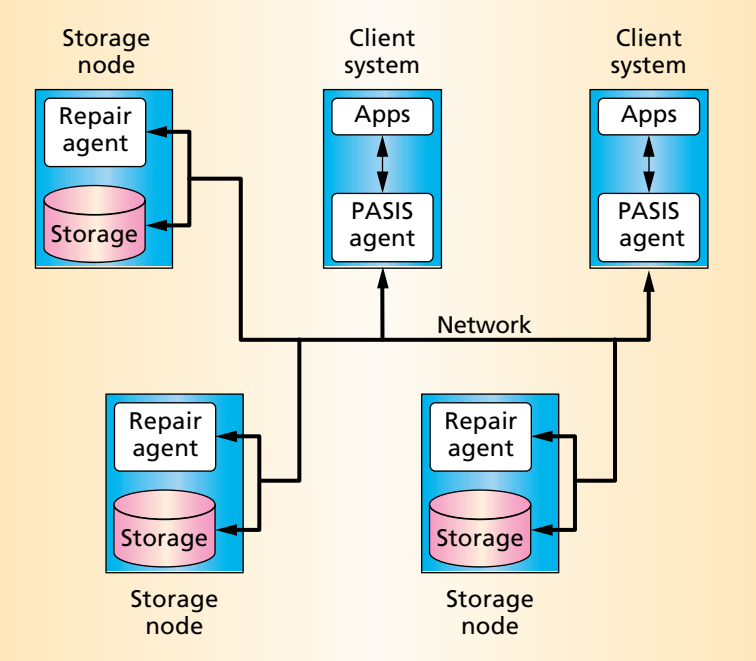

Figure 1. The PASIS architecture. Client systems and storage nodes are attached to the network. Client applications interact with a PASIS storage system through a PASIS agent. Storage devices and repair agents that monitor system status comprise the storage nodes.

but a few systems employ erasure-resilient correction codes, which use less space. Client-side encryption can protect information confidentiality even when storage nodes are compromised.

Threshold schemes—also known as secret-sharing schemes or information dispersal protocols—offer an alternative to these approaches that provides both information confidentiality and availability. These schemes encode, replicate, and divide information into multiple pieces, or shares, that can be stored at different storage nodes. The system can only reconstruct the information when enough shares are available.

For survivable information storage systems, threshold schemes have several advantages over replication and encryption:

- Threshold schemes provide considerable versatility because they divide information into a variable number of shares and require a variable fraction of the shares to reconstruct the information. This flexibility creates a trade-off between information confidentiality and availability. For example, requiring fewer shares for reconstruction means that fewer storage nodes must survive attacks to maintain availability, but also that fewer storage nodes must be compromised for an attacker to learn information.
- Most options within the trade-off spectrum are more space efficient than maintaining replicas on each node.
- Some threshold schemes offer encoding and decoding that are orders of magnitude faster than public-key cryptography.
- Threshold schemes provide information confidentiality from storage nodes without requiring users or client systems to remember or share secret keys.

Dynamic self-maintenance

Over time, all systems need maintenance. The decentralized nature of survivable storage systems makes maintenance difficult and therefore error prone. Truly survivable systems automatically perform some self-maintenance to avoid undesirable conditions. Self-maintenance includes regular monitoring for potential problems, such as failed or compromised nodes, performance bottlenecks, and denial-of-service attacks.

For example, a storage node's failure reduces the number of additional failures that can occur before information is lost. Rebuilding the contents of the lost storage node on a spare replacement restores full redundancy. Another example of self-maintenance is the adjustment of threshold scheme parameters based on observed performance and reliability.

Self-securing storage

Survivable storage systems have great difficulty coping with undesirable requests from legitimate user accounts. These requests can originate from malicious users, rogue programs run by unsuspecting users (for example, e-mail viruses), or intruders exploiting compromised user accounts. Unfortunately, since there is no way for the system to distinguish an intruder from the real user, the system must treat them similarly. Thus, the intruder can read or write anything to which the real user has access. Even after detecting the intrusion, users and system administrators face daunting tasks: determining the damage the intrusion caused and restoring the user's data to a safe state. The "Surviving Malicious User Activities" sidebar explains how self-securing storage offers a partial solution to these problems.

THE PASIS ARCHITECTURE

The PASIS architecture, shown in Figure 1, combines decentralized storage systems, data redundancy and encoding, and dynamic self-maintenance to achieve survivable information storage. A PASIS system uses threshold schemes to spread information across a decentralized collection of storage nodes. Client-side agents communicate with the collection of storage nodes to read and write information, hiding decentralization from the client system. Automated

monitoring and repair agents on storage nodes provide self-maintenance features.

Several current and prior efforts employ similar architectures to achieve survivability. For example, the Intermemory Project[5] and the Eternity Service[6] proposal use decentralization and threshold schemes for long-term availability and integrity of archived write-once digital information. The e-Vault[7] and Delta-4[8] projects do the same for online read-and-write information storage. Delta-4 additionally addresses information confidentiality and other system services, such as authentication. All of these projects have advanced the understanding of these technologies and their roles in survivable storage systems, but such systems have not yet achieved widespread use.

Because these technologies are believed to enhance storage system survivability, we have focused our efforts on designing survivable systems with performance and manageability on a par with simpler, conventional approaches. Only when such systems exist will survivable information systems be widely adopted.

PASIS system components and operation

A PASIS system includes clients and servers. The servers, or storage nodes, provide persistent storage of shares; the clients provide all other aspects of PASIS functionality. Specifically, PASIS clients include agent programs that communicate with collections of PASIS servers to collect necessary shares and combine them using threshold schemes. This approach helps the system scale and simplifies its decentralized trust model. In fact, some system configurations can employ PASIS servers that are ignorant of decentralization and threshold schemes; for example, in our simplest demonstration, the PASIS client uses shared folders on Microsoft's Network Neighborhood as storage nodes. More advanced storage servers are clearly possible.

As with any distributed storage system, PASIS requires a mechanism that translates object names—for example, file names—to storage locations. A directory service maps the names of information objects stored in a PASIS system to the names of the shares that comprise the information object. A share's name has two parts: the name of the storage node on which the share is located and the local name of the share on that storage node. A PASIS file system can embed the information needed for this translation in directory entries. For example, our PASIS implementation of NFS (Network File System) functions in this way.

As the "General Threshold Schemes" sidebar describes, a *p-m-n* threshold scheme breaks information into *n* shares so that any *m* of the shares can reconstruct the information and fewer than *p* shares reveal no information. To service a read request, the PASIS client

- Looks up, in the directory service, the names of the *n* shares that comprise the object.
- Sends read requests to at least *m* of the *n* storage nodes.
- Collects the responses. If it receives fewer than *m* responses, the client retries the failed requests. Alternatively, the client can send read requests to other previously unselected storage nodes. This step continues until the client has collected *m* distinct shares.
- Performs the appropriate threshold operation on the received shares to reconstruct the original information.

Surviving Malicious User Activities

Self-securing storage prevents intruders from undetectably tampering with or permanently deleting stored data. Since there is no way to differentiate intruders from legitimate users, self-securing storage devices consider all requests to be suspect. These self-contained, self-controlling storage devices—for example, enhanced disk drives or PASIS storage nodes—internally version all data and audit all requests for a guaranteed amount of time, such as a week or a month. This provides system administrators with a window of time during which to detect and recover from intrusions. Using this information after an intrusion, administrators can figure out what the intruder did and quickly recover the true user's information. Further, since the device maintains unscrubbable audit logs—that is, they cannot be erased by client-side intruders—security personnel can use the logs to partially identify the propagation of intruder-tainted information around the system.

Experiments and evaluations indicate that self-securing storage is feasible.[1] In particular, performance can be comparable to traditional storage devices, and storage capacities allow reasonable versioning window guarantees. For example, one or more weeks' worth of all changes to all data can be kept on modern storage devices at reasonable cost. Further, with sustained 60 to 100 percent per year capacity growth, the length of this window is expected to grow. With the ability to keep all versions of all data for several weeks, self-securing storage will be an important tool for surviving client and user account intrusions.

References

1. J. Strunk et al., "Design and Implementation of a Self-Securing Storage Device," Tech. Report CMU-CS-00-129, School of Computer Science, Carnegie Mellon Univ., May 2000, to appear in OSDI 2000.

Ex. 1026, Page 3

Performing a write in PASIS requires more steps than performing a read. The write process is similar to a read, but does not complete until at least $n - m + 1$ (or m, whichever is greater) storage nodes have stored their shares. That is, a write must ensure that fewer than m shares have not been overwritten to preclude reading m shares that were not updated. To maintain full availability, all n must be updated.

In addition, consider two clients, A and B, writing to the same object, D, concurrently. Because PASIS storage nodes are independent components of a distributed system, no assumptions should be made about the order in which the storage nodes see the writes. Thus, some storage nodes could have shares for D_A, while others could have shares for D_B. A similar problem arises if a client accidentally or maliciously updates only a subset of the shares.

One approach to handling concurrency problems assumes that a higher-level system addresses them. There are domains in which this approach is feasible. For example, information belonging to a single user and distributed applications that manage their own concurrency can use a storage system that does not guarantee correctness of concurrent writes.

For general-purpose use, a PASIS system must provide a mechanism that guarantees atomicity of operations. One such mechanism, atomic group multicast, guarantees that all correct group members process messages from other correct members in the same order. Atomic group multicast technology has been developed

General Threshold Schemes

A *p-m-n* general threshold scheme breaks information into n shares so that

- every shareholder has one of the shares,
- any m of the shareholders can reconstruct the information, and
- a group of fewer than p shareholders gains no information.

Clearly, m must be less than or equal to n, and p must be less than or equal to m. Figure A illustrates some simple threshold schemes.

Secret-sharing schemes are *m-m-n* threshold schemes that trade off information confidentiality and information availability; the higher the confidentiality guarantee, the more shares required to reconstruct the original information object.[1-3] Secret-sharing schemes can be thought of as a combination of splitting and replication techniques.

Figure B demonstrates how Blakley's secret-sharing scheme works in an m-dimensional space.[1] Secrets (information) are points in the space, and shares are multidimensional planes. Fewer than m shares represent a multidimensional plane that contains the secret. However, since all points in the field being considered are part of the plane, no information is revealed. With m shares, a single point of intersection—the secret—is determined. Shamir's secret-sharing scheme, developed at the same time as Blakley's, is based on interpolating the coefficients of a polynomial by evaluating the polynomial at certain points.[2]

Ramp schemes implement the full range of general *p-m-n* threshold schemes.[4] Ramp schemes operate like secret-sharing schemes up to p shares and like information dispersal schemes with p or more shares.

Threshold schemes can be used instead of cryptographic techniques to guarantee the confidentiality of information, and the two techniques can also be combined. For example, *short secret sharing* encrypts the original information with a random key, stores the encryption key using secret sharing, and stores the encrypted information using information dispersal.[5] Short secret sharing offers a different set of trade-offs between confidentiality and storage requirements than general threshold schemes. The confidentiality of the information stored by short secret sharing hinges on the difficulty of analyzing the information gained by

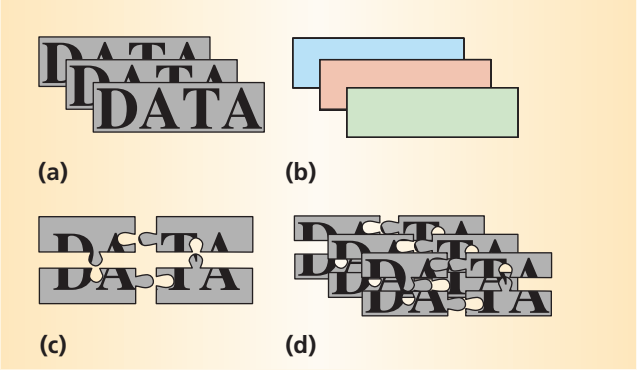

Figure A. Examples of simple threshold schemes: (a) Replication (1-1-n) increases information availability by increasing the storage required by a factor of n. Replication provides no information confidentiality because each share contains an entire copy of the original information object. (b) Splitting (n-n-n) increases information confidentiality by increasing the storage required by a factor of n. Splitting decreases information availability because all n shares must be available. (c) Decimation (1-n-n) divides information objects into n pieces and stores each piece separately. Decimation decreases information availability because all shares must be available; it offers no information-theoretic confidentiality because each share exposes 1/n of the original information. (d) Rabin's[7] information dispersal algorithm (1-m-n) offers a range of information storage solutions that trade off between information availability and required storage. Like decimation, information algorithms do not offer confidentiality.

for several secure distributed systems, such as Rampart[9] and BFS.[10] Unfortunately, atomic group multicast can be expensive in large and faulty environments because the group members often must exchange many rounds of messages to reach agreement.

PASIS architecture characteristics

The PASIS architecture provides better confidentiality, availability, durability, and integrity of information than conventional replication—but at a cost in performance.

To compare the PASIS architecture to a conventional information storage system, consider a PASIS installation with 15 storage nodes that uses a 3-3-6 threshold scheme and uniformly distributes shares among storage nodes. The conventional installation, on the other hand, organizes 15 servers into five server groups, each storing 20 percent of the information on a primary server and two backup replica servers. Table 1 summarizes this comparison.

Confidentiality determines a system's ability to ensure that only authorized clients can access stored information. To breach the conventional system's confidentiality, an intruder only needs to compromise a single storage node that has a replica of the desired object. In a PASIS system, an intruder must compromise several storage nodes to breach confidentiality.

Availability describes a system's ability to serve a specific request. For comparison purposes, assume that each storage node fails independently with a

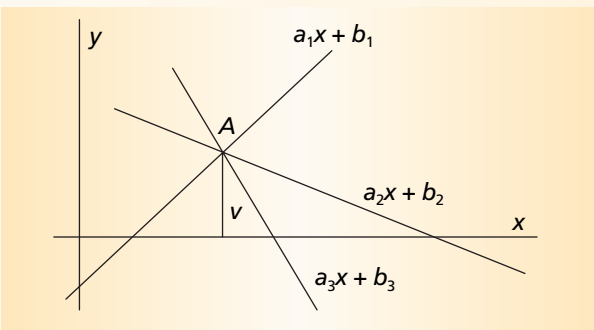

Figure B. Blakley's secret-sharing scheme, with* m = *2 and* n = *3.

collecting shares, because the information gained pertains to the encrypted information object.

An extension to threshold schemes is *cheater detection*.[6] In a threshold scheme that provides cheater detection, shares are constructed in such a fashion that a client reconstructing the original information object can tell, with high probability, whether any shares have been modified. This technique allows strong information-integrity guarantees. Cheater detection can also be implemented using cryptographic techniques, such as adding digests to information before storing it.

Table A shows the properties of some *p-m-n* threshold schemes.

References

1. G. Blakley, "Safeguarding Cryptographic Keys," *Proc. Nat'l Computer Conf.*, American Federation of Information Processing Societies, Montvale, N.J., 1979, pp. 313-317.
2. A. Shamir, "How to Share a Secret," *Comm. ACM,* Nov. 1979, pp. 612-613.
3. E. Karnin, J. Greene, and M. Hellman, "On Secret Sharing Systems," *IEEE Trans. Information Theory,* Jan. 1983, pp. 35-41.
4. A. De Santis and B. Masucci, "Multiple Ramp Schemes," *IEEE Trans. Information Theory,* July 1999, pp. 1720-1728.
5. H. Krawczyk, "Secret Sharing Made Short," *Advances in Cryptology*, D.R. Stinson, ed., Springer-Verlag, Berlin, 1993, pp. 136-146.
6. M. Tompa and H. Woll, "How to Share a Secret with Cheaters," *J. Cryptology,* Feb. 1988, pp. 133-138.
7. M. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *J. ACM,* Apr. 1989, pp. 335-348.

Table A. Properties of some *p-m-n* threshold schemes.

Scheme	Confidentiality	Storage blowup	Examples
Replication	None	n	Full copies
Splitting	Perfect	n	XOR of data
Decimation	Incremental	1	Puzzle pieces
Encryption	Computational	1	Data Encryption Standard
Secret sharing	Perfect	n	Blakley,[1] Shamir,[2]
Information dispersal algorithm	Incremental	n/m	Rabin[7]
Ramp schemes	Perfect, then incremental	$n/(1 + m - p)$	De Santis and Masucci[4]
Short secret sharing	Computational	$n/m + n\varepsilon$	Krawczyk[5]

Ex. 1026, Page 5

Table 1. Comparison of PASIS and conventional systems.

Measure	Node characteristics	PASIS	Conventional
Confidentiality	Percentage of information revealed if one storage node is compromised	0	20 percent
	Percentage of information revealed if three storage nodes are compromised	4.4 percent	Up to 60 percent
Availability	Probability that the system cannot serve a read request if each node fails with 0.001 probability	1.5×10^{-11}	10^{-9}
Durability	Number of nodes that must be destroyed to erase a piece of information	4	3 (a server group)
	Percentage of information erased when the above occurs	1.1 percent	20 percent
Integrity	Nodes that must be compromised to falsely serve a read request	3 (m nodes)	1 (primary server)
	Nodes that must be compromised to modify stored information without authorization	4 (greater($n - m + 1$, m))	1 (primary server)
Required storage (factor by which data stored increases)	Secret sharing	$6\times$ ($n\times$)	$3\times$ ($n\times$)
	Information dispersal	$2\times$ ($n\times/m$)	$3\times$ ($n\times$)
Latency	Reading small objects	Significantly higher latency for PASIS	
	Reading large objects	Similar latencies	

probability of 0.001. With this assumption, the PASIS system has two orders of magnitude higher availability than the conventional system.

Durability is a system's ability to recover information when storage nodes are destroyed. In the conventional system, an intruder must destroy three storage nodes (an entire server group) to maliciously erase information. In the PASIS system, an intruder must destroy four storage nodes to erase information.

Integrity is a system's ability to ensure that it correctly serves requests. To maliciously affect a read request in a PASIS system, an intruder must compromise the m storage nodes serving the read request (assuming a share-verification scheme is in place). In a conventional system, an intruder compromising a primary server can cause it to return arbitrary values to read requests.

Required storage is the extra storage space a system needs beyond a single-copy baseline. In a PASIS system, the storage required depends on the threshold scheme being used. For example, secret sharing requires the same storage as replication, whereas information dispersal requires less storage than replication.

Latency is the delay a system experiences when it serves a request. In a conventional system, a client processes a read or write request by exchanging messages with a single server. In a PASIS system, messages must be exchanged with multiple storage nodes, which can significantly impact performance. However, some threshold schemes require that m servers each provide S/m of a dispersed object's S bytes. For large objects, network and client bandwidth limitations can potentially hide the overhead of contacting m servers.

PASIS architecture performance trade-offs

Threshold schemes can increase an information storage system's confidentiality, availability, and integrity. However, such schemes present trade-offs among information confidentiality, information availability, and storage requirements:

- As n increases, information availability increases (it's more probable that m shares are available), but the storage required for the information increases (more shares are stored) and confidentiality decreases (there are more shares to steal).
- As m increases, the storage required for the information decreases (a share's size is proportional to $1/(1 + m - p)$), but so does its availability (more shares are required to reconstruct the original object). Also, as m increases, each share contains less information; this may increase the number of shares that must be captured before an intruder can reconstruct a useful portion of the original object.
- As p increases, the information system's confidentiality increases, but the storage space required for the dispersed information also increases.

With this flexibility, selecting the most appropriate threshold scheme for a given environment is not trivial.

Clients can also make trade-offs in terms of how they interact with storage nodes that hold shares. Even though only m shares are required to reconstruct an object, a client can over-request shares. That is, a client can send read requests to between m and n storage nodes. By over-requesting, the client reduces the risk of a data storage node's liability to a data storage node being unavailable or slow.

Automatic trade-off selection

For the PASIS architecture to be as effective as possible, it must make the full flexibility of threshold schemes available to clients. We believe this option requires automated selection of appropriate threshold schemes on a per-object basis. This selection should combine object characteristics and observations about the current system environment. For example, a PASIS client could use short secret sharing to store an object larger than a particular size, and conventional secret sharing to store smaller objects. The size that determines which threshold scheme to use could be a function of the object type, current system performance, or both.

As another example, an object marked as archival—for which availability and integrity are the most important storage characteristics—should use an extra-large n. For read/write objects, increased write overhead makes large n values less desirable. Moreover, if the archival object is also marked as public—such as a Web page—the client should ignore confidentiality guarantees when selecting the threshold scheme.

System performance observations can also be used to dynamically improve per-request performance. For example, clients can request shares from the m storage nodes that have responded most quickly to their recent requests. Storage nodes can also help clients make these decisions by providing load information or by asking them to look elsewhere when long response times are expected.

As mentioned earlier, clients can use over-requesting to improve performance. For example, in an ad hoc network with poor message-delivery guarantees, a PASIS client could notice the request loss rate and increase the number of shares requested on a read. On the other hand, in a very busy environment, the excess load on storage nodes inherent to over-requesting can reduce performance.

Proven technologies exist for constructing information storage systems whose availability, confidentiality, and integrity policies can survive component failures and malicious attacks. The main challenges in deploying these systems relate to their engineering. Specifically, we need implementation techniques to help these systems achieve performance and manageability competitive with today's nonsurvivable storage systems. We believe this requirement can be met by aggressively exploiting the flexibility offered by general threshold schemes within the PASIS architecture. More information about PASIS is available at http://PASIS.ices.cmu.edu/. ✸

Acknowledgments

Our PASIS work is supported by DARPA/ISO's Intrusion Tolerant Systems program (Air Force contract number F30602-99-2-0539-AFRL) and by the Pennsylvania Infrastructure Technology Alliance. We thank the many members of the PASIS team, including Ergin Guney, Qi He, Semih Oguz, Joe Ordia, Yaron Rachlin, Xiaofeng Wang, and Mike Vande Weghe, for their efforts and for helping us to develop these ideas. We also thank Mike Vande Weghe and the anonymous reviewers for helping to improve the quality of this article.

References

1. T. Anderson et al., "Serverless Network File Systems," *ACM Trans. Computer Systems,* Feb. 1996, pp. 41-79.
2. G. Gibson et al., "A Cost-Effective, High-Bandwidth Storage Architecture," *Proc. 8th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM Press, New York, 1998, pp. 92-103.
3. E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks," *Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM Press, New York, 1996, pp. 84-92.
4. P. Chen et al., "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys,* June 1994, pp. 145-186.
5. A. Goldberg and P. Yianilos, "Prototype Implementation of Archival Intermemory," *Proc. Advances in Digital Libraries (ADL 98),* IEEE CS Press, Los Alamitos, Calif., 1998, pp. 147-156.
6. R. Anderson, "The Eternity Service," *Proc. PRAGOCRYPT 96*, CTU Publishing House, Prague, 1996.
7. A. Iyengar et al., "Design and Implementation of a Secure Distributed Data Repository," *Proc. 14th IFIP Int'l Information Security Conf. (SEC 98),* ACM Press, New York, 1998.
8. Y. Deswarte, L. Blain, and J. Fabre, "Intrusion Tolerance in Distributed Computing Systems," *Proc. IEEE Symp. Security and Privacy,* IEEE CS Press, Los Alamitos, Calif., 1991, pp. 110-121.
9. M. Reiter, "Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart," *Proc. 2nd ACM Conf. Computer and Communication Security,* ACM Press, New York, 1998, pp. 68-80.
10. M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Operating Systems Review*, ACM Press, New York, 1999, pp. 173-186.

Jay J. Wylie is a PhD student in the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests include distributed systems and security. Wylie received an MS in electrical and computer engineering from Carnegie Mellon University. He is a member of the IEEE. Contact him at jwylie@cmu.edu.

Michael W. Bigrigg is a project scientist with the Institute for Complex Engineered Systems at Carnegie Mellon University. His research interests include software engineering for embedded and reliable information systems, specifically programming languages and systems. He received an MS in computer science from the University of Pittsburgh. Contact him at bigrigg@cmu.edu.

John D. Strunk is a PhD student in the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests include computer system security, storage systems, and distributed systems. Strunk received an MS in electrical and computer engineering from Carnegie Mellon University. He is a member of IEEE. Contact him at jstrunk@andrew.cmu.edu.

Gregory R. Ganger is an assistant professor of electrical and computer engineering at Carnegie Mellon University. His research interests include computer operating systems, storage, security, networking, and distributed systems. Ganger received a PhD in computer science and engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE and the ACM. Contact him at greg.ganger@cmu.edu.

Han Kılıççöte is a research engineer at the Institute for Complex Engineered Systems at Carnegie Mellon University. On leave from CMU, Kılıççöte currently is at Atoga Systems Inc. His research interests include computer networks, distributed systems, and computer security. He received a PhD in civil engineering from Carnegie Mellon University. He is a member of the IEEE and the ACM. Contact him at kiliccote@cmu.edu.

Pradeep K. Khosla is the Philip and Marsha Dowd professor of engineering and robotics and head of the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests include mechatronics, agent-based design and control, software engineering for real-time systems, collaborating robotic systems, gesture-based programming, and distributed information systems. Khosla earned a PhD from Carnegie Mellon University. He is an IEEE fellow. Contact him at pkk@ece.cmu.edu.