Issues and Mechanisms for Trustworthy Systems: Creating Transparent Mistrust

Matt Blaze
Jack Lacy
Thomas London
Mike Reiter

Traditionally, security in distributed systems is viewed as an “extra” that comes only at the expense of convenience, performance, or functionality. Security mechanisms are often provided only at the highest levels of abstraction, and are poorly integrated into whole systems. Consequently, comprehensive security is rarely available except in the most critical applications, where threats are deemed serious enough to warrant the development of special-purpose protection mechanisms. This paper introduces the concept of “transparent mistrust,” an approach that includes security as an underlying part of distributed system interfaces and services. Transparent mistrust relies on security mechanisms that minimize trust in system components without incurring costs commonly associated with security. The scalable mistrust mechanisms described in this paper support a wide range of trust models and security policies in communications networks, file systems, and distributed services.

Introduction

Users of today’s distributed computing systems are asked to place greater trust in their systems than ever before. To an increasing degree, distributed systems are being used to deliver personal and financial services. These range from simple file storage and electronic mail to more ambitious applications, such as electronic commerce, interactive television, and electronic tax services. As a consequence, users entrust such applications with greater amounts of personal, or otherwise private, information, and they rely heavily on these services for critical purposes. In doing so, users trust that these systems will not provide fraudulent services or divulge private data to unintended recipients.

At the same time, the degree to which this trust is warranted must be questioned. Distributed systems are becoming larger, more accessible and complex, and, consequently, less secure. Services such as those described earlier may be implemented within a system of widely distributed databases and client/server architectures that spans the Internet or the entire international telephone network. In systems of such scale and complexity, many opportunities exist for user data to be captured and misused by unintended recipients, and for system services to be compromised by malicious parties.

Previous attempts to address this issue have met with little acceptance, despite the need for strong security mechanisms to reduce how much users must trust these systems. Including security mechanisms in distributed systems seems to decrease system performance and functionality. For example, using cryptography to protect communication from a network eavesdropper can significantly degrade communication performance, unless high-performance cryptographic hardware is available.[1] Moreover, an infrastructure is needed to distribute cryptographic keys, and to help the user bear the burden of properly negotiating and using these keys.

The costs and complexities only become worse when other threats are considered. These complexities are exacerbated in large-scale systems, where multiple administrative domains may exist, all implementing a key distribution infrastructure and independent security policies.

Panel 1. Abbreviations, Acronyms, and Terms

active badge—A personal token, worn on the body, to track location
CFS—Cryptographic File System
cleartext—unencrypted data
DES—Data Encryption Standard
digital cash—an electronic representation of money
DSA—Digital Signature Algorithm
IP—Internet protocol
MD5—Message Digest 5, a public domain message digest algorithm
NIST—National Institute of Standards and Technology
RSA—Rivest-Shamir-Adleman algorithm
signature certificate—a self-authenticating electronic representation of a signature
smart card—credit-card-size, tamper-resistant personal computer

Traditional security mechanisms have avoided these issues by restricting themselves to enforcing limited authorization policies for access to data and services. As systems "scale up," the hierarchical and monolithic models on which these mechanisms are based become difficult to apply; only rarely will all entities in a large-scale system trust a single authority or request a common access policy. It is more natural to control "trust boundaries," across which data and resources flow. These boundaries are both physical and logical; current trust boundaries include network interfaces, remote window clients, and the disks on file servers. Soon, trust boundaries will be associated with digital cash, private cryptographic keys, signature certificates, active badge locations, smart card interfaces, and other critical, if not as well defined, entities.

Cryptographic techniques (data encryption, digital signatures, etc.) offer a promising basis for managing trust, but current systems make cryptography too cumbersome for serious, routine use. Cryptographic protection remains the province of the applications themselves, where it may be difficult to maintain fine-grained control of the data flow across trust boundaries, or to enforce a consistent, reliable security policy. Building trustworthy systems requires cryptographic protection of the underlying interfaces between trust boundaries, injecting mistrust across the layers and interfaces in networks, file systems, naming services, and applications.

Recent technological developments help make wider use of cryptographic techniques feasible. First, computer workstations (the machines with which users directly interact and over which they exert physical control) are becoming fast enough to handle cryptographic protocols routinely without using adjunct hardware. For example, current medium-grade workstations can perform Data Encryption Standard (DES)[2] software encryption at rates of 1 megabit per second (Mbit/s). Second, small, portable cryptographic agents (e.g., "smart cards") are designed to guarantee that secret data never leave a device that remains in the authorized user's physical possession. Finally, practical public-key cryptosystems (e.g., the Rivest-Shamir-Adleman [RSA] algorithm)[3] allow secure data to be exchanged with less prior off-line arrangement, enabling secure transactions to take place outside one's own administrative domain.

The costs traditionally associated with security are not always necessary. Often, they are the result of poor integration of security mechanisms with the larger system. Transparent mistrust is an approach that integrates security into the underlying distributed system interfaces and services. It relies on security mechanisms that minimize trust in system components without incurring the costs usually associated with security. This paper describes several completed and ongoing projects that illustrate this approach in real systems. These projects include mechanisms to implement mistrust of remote file services, communication networks, and a wide range of distributed services, while minimizing the costs to users and implementers of the system.

Security of Stored Data

The file system is perhaps the most tangible trust boundary, because sensitive file data are often vulnerable to attack as long as they exist on a disk, and sometimes even after they have been deleted, through poorly controlled backup copies or analysis of reclaimed disk blocks. Many users think of encryption chiefly as a technique for protecting files, but high-level tools for this purpose are cumbersome to use or vulnerable to a sophisticated adversary. Only rarely are files routinely enciphered, and the protection provided by most user-level cryptographic software is a dangerous illusion. Instead, file data should be protected at a lower level—just before it leaves the trust boundary of hardware under the user's direct control. This control typically ends as soon as data are written to a storage device or transmitted over a network to a remote file server.

The Cryptographic File System. The Cryptographic File System (CFS)[4] is an encrypting file system interface for the UNIX operating system, or those similar to it. (UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/OPEN Corporation.) CFS allows the user to associate cryptographic keys with directories, and runs entirely on the client workstation. It requires no modification to the underlying file system (or file server), and cryptographically protects file contents and some meta-data (file names). Backups and other routine administrative services can take place in the normal manner without encryption keys.

Basically, CFS provides a mechanism to associate "real" UNIX directories (on other file systems) that contain encrypted data with temporary "virtual" names, through which users can read and write cleartext (unencrypted text). These virtual names appear in a separate name space, which is usually called `/crypt`. Users create encrypted directories on regular file systems (e.g., in their home directories) using the `cmkdir` command, which creates the directory and assigns to it a cryptographic "passphrase" that is used to encrypt its contents. To be usable, an encrypted directory must be "attached" using the `cattach` command, which asks for the passphrase and installs an association between the "real" directory and a name under `/crypt`. Cleartext is read and written under the virtual directory in `/crypt`, but the files are stored in encrypted form (with encrypted names) in the real directory. The association is removed with the `cdetach` command, which deletes the cleartext virtual directory under `/crypt`.

CFS protects the contents and names of files by guaranteeing that they are never sent in unencrypted form to a storage system. When CFS is run on a client machine in a distributed system, its protection extends to the file system traffic sent over the network. In effect, it provides the security of end-to-end encryption between the client and server without requiring any encryption on the server's side. Of course, the user must still trust the client system on which CFS is running, because that system manages the keys and cleartext for any encrypted directories currently attached.

File encryption uses a fast software implementation of DES and a novel technique to trade off memory for time. With the latency of only one DES operation, CFS performs roughly the equivalent of encrypting each file block twice.

Informal benchmarks suggest that, on a typical workstation with a moderate local disk and cache, performance is probably reduced by less than a factor of two compared to the underlying file system. In contrast, command-based application-level encryption reduces throughput by about a factor of five. The reason for the difference in performance between CFS and application encryption is that, as a file system, CFS avoids many encryptions and decryptions by caching, and CFS does not create new processes for each file access. In day-to-day use, CFS performance and semantics are almost completely transparent. About 500 computer installations in the U.S. and Canada, internal and external to AT&T, are using CFS.

File System Key Management. Key management for encrypted files is fundamentally different from that used for cryptographic communication. In a secure communication system, keys must be distributed and synchronized geographically. They often serve the dual purpose of authenticating identity and protecting against eavesdroppers. Because the architecture for distributing communication keys is closely tied to trust relationships within the system, practical key distribution protocols must be carefully engineered to balance reliability, security, and performance.

In a file system, on the other hand, there is little need to distribute keys geographically; most protected files are encrypted and decrypted at the same location (and by the same users). Authentication of identity is a less serious issue, because access is implicitly controlled by knowledge of the key itself. Cryptographic techniques can also be used to detect unauthorized tampering with file data. File systems still present a significant, if differently formulated, key management problem, in that keys can be said to be distributed *temporally*. The corresponding keys must be available during both encryption and decryption. A file encryption key has a much longer lifetime than its communication counterpart. If a key is lost or unavailable, the files encrypted with it are rendered useless. Unfortunately, this condition is not usually detected until it is too late.

Arguably, because of difficulties associated with key management, sensitive files are rarely encrypted in practice, even when encryption tools are available. This

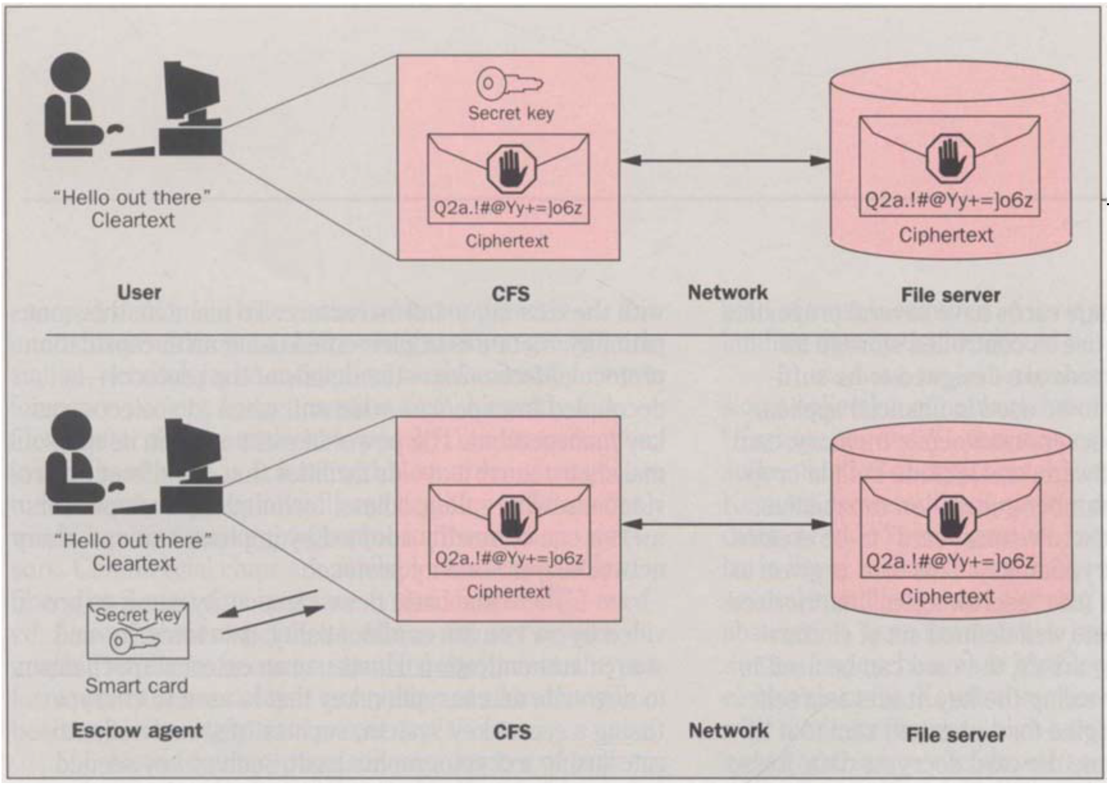

Figure 1. Flow of data in the (a) CFS user system and (b) CFS escrow agent system.

is especially true in critical business environments, where ensuring the availability of data to authorized users is at least as important as ensuring its unavailability to everyone else. Sometimes, files are protected with weak ciphers so that encrypted data can be recovered using sufficient computing resources.

In the context of organizational information systems, cryptographic file protection presents several problems not addressed by traditional, communication-oriented, key management schemes. These problems are not only technical in nature (e.g., providing mechanisms for ensuring that keys are available when and where authorized), but also managerial and social (balancing secrecy and privacy against emergency access requirements). These often conflicting goals can be reconciled, in part, by carefully controlling key management services with explicit, auditable trust relationships that are integrated into the underlying file system security architecture.

Numerous approaches to providing a bypass mechanism for encrypted files are being examined. In one scheme, key information is disassembled and distributed among various trusted "escrow agents" using several techniques: The key could be placed in a cryptographic "box" that requires more than one escrow key to unlock; incomplete keys—based, perhaps, on number theoretic properties—could be distributed. To access the key, a specified number of escrow agents would have to agree.

Another approach provides a controlled mechanism that enables users to deposit copies of their keys to be used for emergencies. The keys for crucial files could thereby be "escrowed" with a trusted caretaker who would only reveal them when certain conditions are met, such as when encrypted business data are needed after the death of the legitimate key holder. Conceptually, keys might be delivered within sealed "envelopes." When a set of files is no longer critical, the envelope containing its keys could be returned to its originator, who could verify the integrity of the seal and destroy the keys, preventing future access to outdated, but still private, data.

Unfortunately, this is difficult to accomplish with software. As a simple solution, the key holder can write down the key, put it in a sealed envelope, and leave it with a trusted caretaker. This is vulnerable to mistakes, however, because no inherent mechanism exists to ensure that the escrowed key is the same as the real one. The security of the scheme also depends entirely on the honesty of the caretaker and the tamper-resistance of the envelope. An electronic analog to the sealed envelope can be implemented by encrypting the key with a "caretaker" key, perhaps using public-key encryption techniques. If this becomes an automatic part of key generation, key holders will be able to avoid the problems associated with transcription mistakes. The scheme, however, will still depend entirely on the caretaker's honesty (more so without the sealed envelope). If no single caretaker can be trusted, the key will have to be encrypted several times, with more than one caretaker's key split among several escrow agents (in the manner of the U.S. Escrowed Encryption Standard), or encrypted using a group-oriented public-key protocol.

Cryptographic smart cards can be used to implement more carefully controlled, fully revocable file-system key escrow.[5] Smart cards have several properties that lend themselves to use as controlled storage for escrowed keys. These cards are designed to be sufficiently tamper-resistant to be used in financial applications, have a controlled-access nonvolatile memory, can run general-purpose software, and include built-in cryptographic and random-number-generation capabilities.

CFS allows an "escrow smart card" to be created along with each file encryption key. This card is given to a designated third party (the "escrow agent") authorized to use the key under some well-defined set of circumstances. If an emergency arises, the card can be used to decrypt files without revealing the key. It acts as a self-contained decryption engine for ciphertext sent to it by the escrow agent. Anytime the card decrypts data, it also logs the access in its secure storage. Later, when the escrow period is terminated, or when an audit is to be performed, the user can query the card to determine whether the escrow agent has used it (see Figure 1).

CFS implements key escrow when the encrypted directory is created with `cmkdir`. When the keys are initially assigned, or whenever escrowed access is required, the machine running CFS must have a smart card reader–writer attached. (No smart card reader is required for day-to-day use on encrypted files.) The smart card must have a small store of secure memory, the ability to run simple programs securely, and a secret-key cryptographic engine.

Security of Communication Networks

The network is one of the most vulnerable trust boundaries in a distributed system, especially when the underlying physical and link layers are based on broadcasting (Ethernet, wireless networks, etc.). The explosively increasing size of the Internet—coupled with the wide range of services and applications that it supports and its lack of existing security features—makes Internet protocol (IP)-based networks good candidates for demonstrating network-layer protection and trust.

Although a number of efforts have been made to provide security services at some layer in the network hierarchy,[6-8] no existing protocol is completely satisfactory for large-scale Internet-based deployment. Our network-layer security protocol for the IP protocol suite, `swIPe`,[9,10] provides authentication, integrity, and confidentiality of IP datagrams, and is completely compatible with the existing IP infrastructure. To maintain this compatibility, `swIPe` is implemented using an encapsulation protocol. *Mechanism*—the details of the protocol—is decoupled from *policy*—what and when to protect—and key management. The power of `swIPe` lies in its minimalist structure; it avoids facilities that could best be provided elsewhere. Regardless, techniques developed for `swIPe` can be readily adopted by implementations of any network-layer security protocol.

The three basic network security services provided by `swIPe` are confidentiality, data integrity, and source authentication. Hosts use an external mechanism to negotiate an encryption key that is used to encrypt (using a secret-key system, such as DES) and authenticate (using a cryptographic hash, such as key-seeded MD5) IP datagrams. The encryption key is bound to an IP address pair; if the encryption key negotiation is secure, the network address can be used as a reliable, secure identifier for incoming and outgoing data.

By encapsulating `swIPe` datagrams within IP datagrams of a new protocol type, `swIPe` complements IP functionality, adding necessary security features that do not alter the structure of IP itself. These datagrams carry the payload of the original IP datagram, enough header information to reconstruct the original packet at the remote end, and any additional security information that may be needed.

This approach has a number of advantages. First, because datagrams are encapsulated within other IP datagrams, they can traverse parts of the network that know nothing about `swIPe`. Second, because the transport and higher-layer protocols never see `swIPe` datagrams, they can continue to use the existing network infrastructure and interfaces. Third, encapsulation and decapsulation can take place wherever IP datagrams are processed (i.e., either hosts or routers), enabling `swIPe` to implement both end-to-end and intermediate-hop security. Finally, a wide variety of security policies can be carried out simply by controlling when to encapsulate outgoing datagrams and when to accept incoming datagrams (which may, or may not, be `swIPe` encapsulated).

This prototype implementation runs quickly, with its performance bound primarily to the cost of the underlying encryption code. On a modern workstation, `swIPe` encapsulation or decapsulation adds about 100 microseconds (μs) to packet processing time; the DES encryption of a 100-byte packet adds another 900 μs. For interactive traffic on an off-the-shelf medium-grade workstation, network layer encryption carries virtually no performance penalty. Large transfers (e.g., bit maps and files) suffer a more noticeable penalty, but the reduction in total bandwidth is at least within an order of magnitude. Clearly, high-bandwidth applications and services would benefit from cryptographic hardware co-processors. Commercial chips are available that can handle fiber-distributed data interface (FDDI) data rates, if needed. As workstations get faster, and as adjunct cryptographic hardware becomes standard equipment, the performance of network-layer encryption of all traffic will become increasingly transparent.

Network encryption solves a number of problems that are hard to solve elsewhere. Higher-layer solutions like Kerberos[11] require modification of the applications and services that use them and, again, make it difficult to enforce a consistent security policy. Lower-layer security typically requires special processing (and trust) as traffic crosses each new network or administrative domain, often moving control away from the sources and destinations of the data.

Key management and policy configuration raise interesting issues. Clearly, large-scale network security requires some method of distributing the keys used to communicate. Once a key is established, a host still must decide whether to trust the other machine and determine what types of data to accept from it. Simple tuning of the packet-acceptance filtering mechanism allows it to accept a wide range of security policies; still unclear, however, are the implications of accepting, rejecting, or requiring encryption and authentication for a particular packet type. In particular, an important open question is how network security interacts with security requirements at higher levels of abstraction. `swIPe` is beginning to provide some practical experience with these issues.

Infrastructure

One important problem in modern cryptosystems is the distribution of public keys and the negotiation of initial cryptographic exchanges. Reliable, robust, and efficient binding among network address, user names, and public keys is required for large-scale implementation of mistrust in a variety of interfaces. It is not obvious how these bindings should be performed, what kind of trust is required of their sources, and what the binding data structure should be.

Keynote: Secure Interfaces and Services. We envision a collection of network-based services—the "Keynote services"—that support the establishment and authentication of cryptographic secure communication. Keynote services distribute key data, such as certificates, session keys, signature keys, etc., and, while similar in flavor to a traditional name server, they differ in several significant ways. Serving a variety of layers of abstraction, from the network (such as authenticating a `swIPe` key exchange) to high-level applications (secure electronic mail), Keynote services provide applications with bindings that map high-level identifiers (names, e-mail addresses, userids, etc.) to low-level security identifiers (keys). The security requirements of Keynote services are an open issue, because they must be trustworthy (i.e., capable of providing reliable data) without necessarily being trusted (i.e., knowing a client's secret data or being able to violate client security requirements).

Essentially, Keynote services provide a standard interface for applications to establish a secure channel with some other party, service, or object (such as a signed file). The "Black Pages", the application interface to Keynote services, provide a high-level abstraction for obtaining, for example, a session key for a given remote entity under a specific policy. A collection of common local Keynote services enforce the policy, negotiate keys with remote entities, maintain local key databases, and perform name service bindings.

Keynote services are based on the premise that it is much more difficult (and dangerous) to obtain security data (such as session keys and policy) than it is for an application to make use of that data. By providing a loosely coupled interface to standard services, Keynote services free applications from the need to make decisions about global security policy or to negotiate complex protocols. See Figure 2 for the architecture of Keynote services.

CryptoLib: Cryptography in Software. Many cryptographic protocols have been devised to provide varying degrees of privacy and security, using both symmetric (i.e., secret-key) and asymmetric (i.e., public-key) cryptosystems. Efficient, secure implementations of these cryptosystems are generally possible only in hardware, or at least coded in nonportable assembly language for

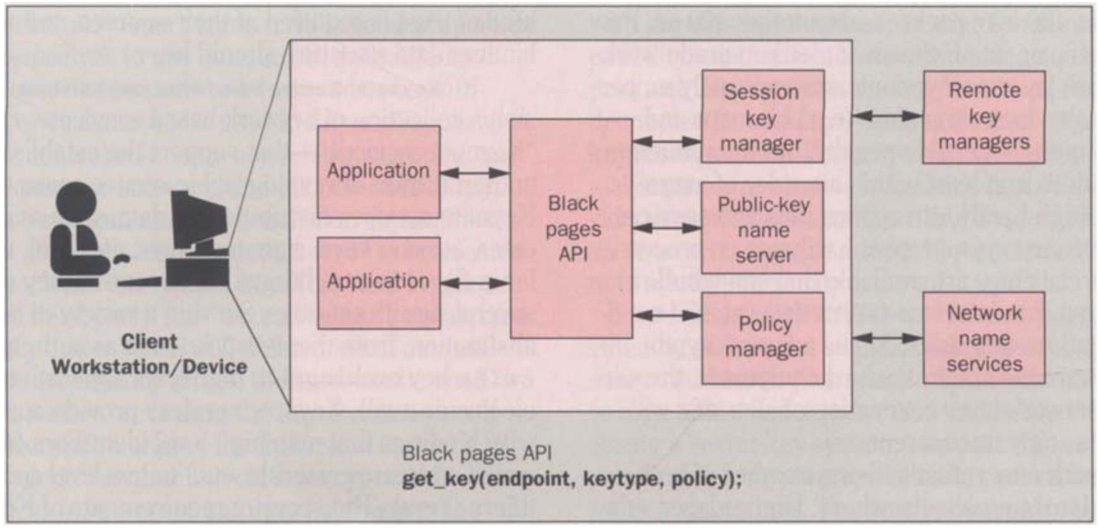

Figure 2. The architecture of Keynote services, which provide applications with a standard interface to security services.

particular processors. Hardware implementations are hardly ubiquitous, however, and the programming overhead of assembly implementations is prohibitive. A portable, efficient software implementation is needed if designers are to have the set of tools necessary to research privacy and security issues and to implement related protocols in the contexts for which they are intended.

CryptoLib[12] is a library of tools needed to implement several symmetric and asymmetric cryptosystems. Although it is written entirely in the C programming language, it uses assembly language for greater efficiency of 32-bit multiplication functions. CryptoLib implements arbitrary-precision arithmetic and several cryptographic algorithms, including the DES,[2] RSA,[3] ElGamal,[13] Digital Signature Algorithm (DSA),[14] and the MD5[15] and National Institute of Standards and Technology (NIST) secure hash[16] functions.

Table I shows the performance of some of the algorithms in CryptoLib, on a 486/50 computer running the UNIX Operating System. If we compare these numbers to the speeds of the fastest hardware implementations of the functions described (e.g., 1 gigabit per second for DES[17] and 1 megabit per second for RSA, with a 512-bit modulus[18]), our software speeds fall short of that achievable in hardware. However, while hardware implementations impose little overhead, they are not cheap and cannot be expected to become ubiquitous. The performance of our DES software implementation is useful in applications such as real-time encryption of Integrated Services Digital Network (ISDN) B channels (64 kbit/s), 64-kbit/s digital speech, and file encryption, as described earlier.

Public-key algorithms implemented in CryptoLib are sufficient for non-real-time applications in which delays of a few seconds are tolerable. For example, key exchange using a public-key algorithm incurs public-key operations only when a communication session is established, not while it is in progress. Whether or not software speeds for public-key operations will be sufficient for extremely time-sensitive applications remains an open question. However, significantly greater execution speeds are possible for modular arithmetic software executing on digital signal processors.[19]

Rampart: Protocols for Reliable Services. The Keynote project described earlier facilitates secure communications and transactions in heterogeneous, large-scale distributed systems. In part, Keynote services were developed to distribute the cryptographic keys needed for secure communication. All known techniques for achieving key distribution require the assistance of some "trusted" services (e.g., some form of an authentication service) to distribute the keys. For two reasons, the number of trusted services will increase in large-scale systems. First, services in different administrative domains may be used to establish communication among participants in different domains. Second, more

Table I. CryptoLib Performance, 486/50

Activity	Time
Generate 512-bit prime	24 s
Create modular exponentiation (512 bits)	400 ms
Generate RSA signature (512 bits)	150 ms
Generate DSA signature (1024 bits)	233 ms
Verify DSA signature (1024 bits)	433 ms
DES encryption	1.4 Mbit/s

types of services (e.g., naming and location services) may be needed to carry out addressing and cryptographic negotiation.

Earlier, we described a technique that reduces the degree of trust required to use a remote file service. Unfortunately, it is not possible to use similar techniques to reduce the degree to which we trust the types of services that are typically involved in key distribution and that form the foundation of Keynote services. Unlike a remote file service, Keynote services are not passive, that is, they do not simply provide storage for clients. Rather, they may create or otherwise use some of the data they hold and distribute. So, it is not possible to keep server data encrypted (with keys unknown to the servers). A second factor to consider is that CFS focuses primarily on ensuring the *secrecy* of the data stored in a file system. In a name service or public-key authentication service, the *availability* of the service and the *integrity* of the information it distributes are more crucial than other factors (e.g., public keys and addresses typically are not kept secret).

Building services that are both highly available and highly secure is difficult because of the conflicting goals of availability and security. The only generally feasible technique for making data and services highly available, namely replicating them, also makes them more difficult to physically protect from an attacker.[20-22] A reasonable approach to balancing this tradeoff in services, such as those in Keynote, is to replicate the services for high availability, but to compensate for the increased difficulty of protecting them by making them tolerant of the malicious penetration of some of their component servers. Even the corruption of up to a threshold number of servers should not result in a denial of service to clients or cause the service to provide incorrect replies to clients.

Techniques for building such highly reliable services have been known for some time.[23] The general idea, called *state machine replication*, is to implement a service with many replicas of a single *deterministic* server, that is, a server whose outputs are determined exclusively by its initial state and the sequence of requests it processes. In this technique, each server is initialized to the same state, and client requests are issued to the servers using a protocol that ensures that all servers process the same requests in the same order. This guarantees that all correct servers produce the same response for any given request. If a client accepts the response that was output by a majority of the servers, the client will accept a correct response from the service, provided that a majority of the servers are correct. Even if a minority of the servers are corrupted by a malicious attacker, the attacker will be unable to affect the integrity or availability of the service.

AT&T researchers are building a toolkit of protocols, called Rampart, to help construct highly reliable services with the techniques of state machine replication, including several security extensions.[24] Clients will be able to use the protocols in this toolkit to submit requests to the service; the toolkit's voting protocols can be used to determine the correct outputs of the service. The foundational protocols of Rampart have been designed and implemented,[25,26] and an initial prototype of the system is almost complete.

Conclusions

Creating transparent mechanisms and infrastructure for large-scale "mistrust" in networked computing is a multifaceted problem. No single "toolkit" can capture the broad range of approaches required to address it. A better understanding of trust, however, and the tools to implement it, are vital to enabling safe, effective use of the emerging information infrastructure.

This paper has examined several aspects of trust and security architecture. Security mechanisms best reside at the lowest levels at which they can do their jobs. The client file system appears to be the right place to protect stored data; the alternatives are inconvenient, inefficient, leave windows of vulnerability, or have the wrong degree of protection. Communication traffic, traditionally protected either at the link layer or as an end-to-end service of the applications themselves, is often more easily and safely protected as a service of the network interface.

The mere availability of secure computing tools is not enough; the infrastructure to support and deploy security on a large scale is equally critical. Software implementations of secure protocols and encryption, once thought too impractical or insecure, are rapidly displacing hardware "black boxes" as standard components of secure systems. Efficient, correct software implementations of cryptographic functions are needed to make large-scale secure computing feasible on a wide range of platforms. Key distribution and management, often left as an afterthought, are emerging as a critical bottleneck; applications are often left without a portable, standard interface for communicating security information among themselves. Standard interfaces to key management services, and strong protocols for building these services in a reliable manner, will lead to much wider acceptance of security as a basic, required feature of emerging applications.

References

1. M. Satyanarayanan, "Integrating security in a large distributed system," *ACM Transactions on Computer Systems*, August 1990, Vol. 7, No. 3, pp. 247-280.
2. *Data Encryption Standard*, National Bureau of Standards, Federal Information Processing Standards Publication 46, Government Printing Office, Washington, D. C., 1977.
3. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, February 1978, pp. 120-126.
4. M. Blaze, "A cryptographic file system for UNIX," *First ACM Conference on Communications and Computing Security*, Fairfax, Virginia, November 1993.
5. M. Blaze, "Key Management in an Encrypting File System," *Proc. Summer 1994 USENIX Tech. Conf.*, June 1994.
6. National Institute of Standards and Technology, NISTIR 90-4250: *Secure Data Network System (SDNS) Network, Transport, and Message Security Protocols*, February 1990.
7. ISO/IEC JTC1/SC6, ISO-IEC DIS 11577 — Information Technology — Telecommunications and Information Exchange Between Systems — Network Layer Security Protocol, November 29, 1992.
8. D. P. Anderson, et al., "A Protocol for Secure Communication in Large Distributed Systems," *Technical Report UCB/CSD 87/342*, University of California, Berkeley, February 1987.
9. J. Ioannidis and M. Blaze, "Architecture and Implementation of Network-Layer Security Under UNIX," *Proceeding of the USENIX Security Workshop*, Santa Clara, California, October 1993.
10. J. Ioannidis and M. Blaze, "The swIPe IP Security Protocol," Internet Draft, August 1993.
11. J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An authentication service for open network systems," *Proceeding of the USENIX Winter 1988 Technical Conference*, February 1988.
12. J. B. Lacy, D. P. Mitchell, and W. M. Schell, "CryptoLib: cryptography in software," *USENIX Security Symposium IV Proceedings*, October 1993, pp. 1-17.
13. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, IT-31(4), July 1985, pp. 469-472.
14. *National Institute of Standards and Technology (NIST) Digital Signature Standard, Federal Information Processing Standards Publication XX,* National Technical Information Service, U.S. Dept. of Commerce, DRAFT, February 1, 1993.
15. R. L. Rivest, "The MD5 Message Digest Algorithm," *Request for Comments (RFC) 1321*, April 1992.
16. *National Institute of Standards and Technology (NIST) Secure Hash Standard, Federal Information Processing Standards Publication 180,* National Technical Information Service, U.S. Dept. of Commerce, DRAFT, February 1, 1993.
17. H. Eberle, "A high-speed DES implementation for network applications," *Research Report 90*, DEC Systems Research Center, September 1992.
18. M. Shand and J. Vuillemin, "Fast implementations of RSA cryptography," *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, July 1993.
19. S. R. Dusse, "A cryptographic library for the Motorola DSP56000," *Advances in Cryptology—EUROCrypt '90*, Lecture Notes in Computer Science 473, I. B. Damgard, ed., 1991, pp. 230-244.
20. R. Turn and J. Habibi, "On the interactions of security and fault-tolerance," *Proceedings of the 9th NBS/NCSC National Computer Security Conference*, September 1986, pp. 138-142.
21. M. P. Herlihy and J. D. Tygar, "How to make replicated data secure," *Advances in Cryptology—CryptO '87 Proceedings*, Lecture Notes in Computer Science 293, 1988, pp. 379-391.
22. B. Lampson, et al., "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems*, Vol. 10, No. 4, November 1992, pp. 265-310.
23. F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, Vol. 22, No. 4, December 1990, pp. 299-319.
24. M. K. Reiter and K. P. Birman, "How to securely replicate services," *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 3, May 1994, pp. 986-1009.
25. M. K. Reiter, "A secure group membership protocol," *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, May 1994, pp. 176-189.
26. M. K. Reiter, "Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart," *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, November 1994.

(Manuscript approved July 1994)

Matt Blaze *is a member of technical staff in the Computing Architectures Research Department at AT&T Bell Laboratories in Holmdel, New Jersey. His current research interests include cryptography security, and scale in distributed systems. He joined AT&T in 1992, after receiving a B.S. from Hunter College, New York City, an M.S. from Columbia University,* *New York City, and a Ph.D. from Princeton University, New Jersey, all in computer science.*

Jack Lacy *is a member of technical staff in the Information Systems Research Laboratory at AT&T Bell Laboratories in Murray Hill, New Jersey. His current research interests include infrastructure necessary for the routine use of public-key cryptography. Mr. Lacy received a B.S. in physics from the University of Alabama, Birmingham, an M.S. in physics from the University of Wisconsin, Madison, and an M.S. in computer science from New York University, New York City. He joined AT&T in 1982.*

Thomas London *is the head of the Computing Architectures Research Department at AT&T Bell Laboratories in Holmdel, New Jersey. He is conducting work in fast implementations of bit-oriented compression algorithms and the application of modern cryptography to the design of practical secure systems. He has previously worked on computing topics including information retrieval, operating systems, compilers, multiprocessor computer systems, communication services, and security. Mr. London joined AT&T in 1976 after receiving a B.A. in mathematics from the University of Pennsylvania, Philadelphia, and an M.S. and Ph.D. in computer science from Cornell University, Ithaca, New York.*

Mike Reiter *is a member of technical staff in the Computing Architectures Research Department at AT&T Bell Laboratories in Holmdel, New Jersey. His research interests include security and fault tolerance in distributed computing systems. Mr. Reiter joined AT&T in 1993 after receiving a B.S. in mathematical sciences from the University of North Carolina, Chapel Hill, and an M.S. and Ph.D. in computer science from Cornell University, Ithaca, New York.*