Michael K. Reiter

Distributing Trust
with the Rampart Toolkit

The Rampart group communication protocols are designed to distribute trust among a group of nodes in a distributed system—so while individual nodes need not be fully trusted, the group can be.

Many mechanisms for enforcing security policy in distributed computer systems rely on trusted nodes, that is, computers and resident software whose correct functioning is essential to implementing the policy. In this article, we describe our research on distributing trust among a group of nodes, so while no single node need be fully trusted, the function performed by the group can be. Central to this effort is a toolkit of group communication protocols called Rampart we developed to simplify construction of such groups.

An overarching goal of computer security mechanisms is to limit the extent to which each component of a system must be trusted to implement a desired security policy. Typically, this goal leads to system designs in which components are labeled

Rampart supports distributed process groups, *a programming paradigm shown to simplify construction of distributed programs tolerant of simple failures.*

"trusted" or "untrusted," the trusted ones constituting a small kernel of hardware and software on which the proper enforcement of the security policy relies. In distributed systems, these trusted components often take the form of localized and physically protected nodes from which the security of the larger system is leveraged. Network firewalls, file servers, authentication servers [15], and certification authorities [10] are all examples of enforcing security policy by leveraging trust in a few nodes.

Distributed systems also offer another means for arranging trust, so it is necessary to trust only some reasonably large fraction of a group of nodes to behave correctly. That is, no node in the group is trusted completely, and even the arbitrary behavior (*Byzantine failure* [11]) of a few nodes (e.g., due to physical capture, electronic penetration, or insider malfeasance) is survivable. Provided that sufficiently few nodes misbehave, the group, acting in concert, can be trusted as a whole. Distributing trust in this way has obvious security and fault-tolerance advantages over the placing-all-your-eggs-in-one-basket alternative.

There has been substantial research on techniques for distributing trust; for surveys of particular classes of techniques, see [3, 8, 20, 21]. In some cases, the results of this research have been dramatic. For example, it has been shown that a group of nodes, each with a private value, can reliably compute any function of those values so that no information about any correct node's input is revealed to other nodes beyond what can be computed from the function's output, despite even the collaboration of a minority of faulty nodes [9]. Distributing trust has also been applied in such cases as clock synchronization [13], file storage, and security administration [4].

Despite a few limited successes, however, most research in this area has not affected distributed systems practice. This is largely due to the inherent complexity of coordinating the actions of many nodes, some of which may behave in unexpected ways. Distributed coordination in the presence of failures is difficult—even more so when the failures may be intentionally arbitrary—and is beyond the reach of the average programmer. Infrastructure and tools for distributing trust are essential if we are to see distributed trust become viable for common use.

The Rampart System

At AT&T Bell Laboratories, we are engaged in an ongoing research effort to overcome these barriers to distributing trust in practical distributed systems. To this end, we developed Rampart, a toolkit of protocols designed to simplify the task of coordinating correctly operating nodes in a dynamic environment characterized by simple (e.g., crash) and arbitrary node failures, node recoveries, and communication failures. Rampart provides support for distributed *process groups*, a programming paradigm shown to simplify construction of distributed programs tolerant of simple failures [1]. Rampart supports communication primitives by which group members can multicast messages to the group, and membership primitives by which a process can be removed from or added to the group if it (or the node on which it executes) fails or recovers, respectively.

The semantics of Rampart process groups borrow much from the notion of *virtual synchrony* introduced by the Isis system [1] and further developed in the Horus [2], Transis [5], and Totem [14] systems. Intuitively, a virtually synchronous process group is one in which each multicast and membership change appears to be executed as a globally indivisible and instantaneous event. Each multicast and membership change is delivered to all group members, and each is ordered identically relative to other events at all group members. These semantics enable a programmer to implement a distributed application without concern for many of the complexities introduced by distribution in a system in which failures can occur [1].

Though Rampart process groups offer semantics similar to the virtually synchronous process groups of many simple-fault-tolerant systems, their implementation is more complex due to the possibility of arbitrary member failures admitted by our system model. For example, implicit in the semantics of a group multicast in the virtual synchrony model is that all group members receive the same multicast message. An arbitrarily faulty multicast initiator may, however, send different messages to different group members in a single "multicast." For such reasons, the Rampart process group implementation employs *agreement protocols* by which correct group members reach agreement on the contents of multicasts and the ordering of events.

The agreement protocols underlying Rampart

process groups are a *group membership protocol* [18] and *reliable* and *atomic* (totally ordered) *group multicast protocols* [16, 17]. The role of the group membership protocol is to enforce agreement among correct processes on the group composition, despite communication failures and (simple or arbitrary) process failures. The reliable and atomic group multicast protocols then enforce agreement on the contents of each multicast to the group (even from an arbitrarily faulty member) and order multicasts relative to group membership changes and other multicasts. These protocols, and thus a Rampart process group, are suitable for use in asynchronous systems; that is, they require no upper bounds on network delays or relative clock drifts among members. Moreover, they can tolerate the arbitrary failure of fewer than one-third of each instance of the group membership.

The semantics offered by Rampart process groups can simplify distributing trust in systems, and indeed, prior work has already placed reliable group multicast (*Byzantine agreement* [11]) and atomic group multicast at the heart of many techniques for distributing trust (see [8, 20]). For example, *state machine replication* [20] is a general technique for constructing services that continue to operate correctly despite the simple or arbitrary failure of component servers. In this technique, a service is implemented using multiple identical deterministic servers, initialized to the same state. Clients issue requests to the service using an atomic multicast protocol, so that all correct servers receive and process the same sequence of requests and thus return the same reply for each request. The client accepts the response returned by a majority of the servers, ensuring that the outputs of a faulty minority of the servers are ignored.

Applications

In conjunction with developing Rampart, we are focusing on applications it can enable. Two efforts, pursued with colleagues at Bell Labs, have resulted in prototype services that illustrate Rampart's potential utility, while also being of interest themselves. The two applications are a service for performing sealed-bid auctions [6] and a cryptographic key management service [19]. Both demonstrate techniques for distributing trust among many servers so that the arbitrary failure of fewer than one-third of the servers risks neither the integrity or the availability of the service, nor the secrecy of the confidential information it holds.

The first service illustrates how auctions, which have already appeared on the Internet [12], can be implemented in conjunction with next-generation secure commerce techniques to promote direct competition in electronic buying and selling. Our service offers an interface through which bidders can submit secret monetary bids for an advertised item. Once the bidding period has ended, the auction service opens the bids, determines the winning bid, and provides the winning bidder with a ticket for claiming the item. The service provides strong protection for both the auction authority and correct bidders, despite the arbitrary failure of any number of bidders and fewer than one-third of the servers comprising the auction service. Specifically, it is guaranteed that bids of correct bidders are not revealed until after the bidding period has ended; the auction authority collects payment for the winning bid; losing bidders forfeit no money; only the winning bidder can collect the item bid upon; and, if desired, bidders can remain anonymous.

The second application is a service for managing cryptographic keys in open networks, called Ω ("Omega"). Ω supports traditional interfaces for managing public keys, including interfaces for a client to register a public key for a principal (e.g., person, computer), retrieve a public key for a principal, and revoke a public key on behalf of a principal. In addition, motivated by the need for key backup to ensure that critical information can be decrypted, Ω provides interfaces for escrowing the private keys corresponding to the public keys it distributes. This mechanism allows an escrowed private key to be reconstructed if, for example, it is lost by its owner, or to be used to decrypt data selectively in a protected and auditable way, such as in emergency situations when the owner of the private key is not available. This mechanism can also be tailored to support law enforcement access to encrypted communications. Ω ensures the integrity and availability of its functions, and the confidentiality of the private keys it escrows, despite the arbitrary failure of fewer than one-third of the servers comprising the service.

Both our auction service and Ω are built using the technique of state machine replication (described earlier) to ensure that the failure of servers does not result in clients receiving incorrect responses to requests. Moreover, in both of these

Rampart has been adapted for two notable applications: *sealed-bid auctions and cryptographic key management*

Rampart greatly simplifies the task of distributing trust *among multiple nodes in a system*

services, further measures are required to protect the secrecy of information stored at the service in the event of server penetrations. In the case of the auction service, that private information is digital cash each bidder escrowed at the service to back its bid. In the case of Ω, that information is private keys that are escrowed at the service. The requirement to protect this information from arbitrarily faulty servers implies that the information can never be collected at a single server, and thus the validation and use of this information must be performed as distributed computations. In Ω, this is done with known techniques [3]; our auction service, however, required new ones [7].

Conclusion

At the time of this writing, Rampart is a research prototype and the subject of ongoing work. While it is too early to deem Rampart a success, our initial experiences with the system indicate it can greatly simplify the task of distributing trust among multiple nodes in a system, and that it performs sufficiently well to support a wide range of applications. Our plans include efforts to extend the protocols to accommodate new failure models and to improve system performance. In addition, we are preparing Rampart for release to the scientific community free of charge to encourage peer review of the system and experimentation with challenging applications. For more information, write to reiter@research.att.com.

References

1. Birman, K.P. The process group approach to reliable distributed computing. *Commun. ACM 36*, 12 (Dec. 1993) 37–53.
2. Birman, K.P., Maffeis, S., and van Renesse, R. Software support for distributed modularity in Horus. *Commun. ACM 39,* 4 (April 1996).
3. Desmedt, Y. Threshold cryptography. *European Transactions on Telecommunications and Related Technologies 5*, 4 (July 1994) 449–457.
4. Deswarte, Y., Blain, L., and Fabre, J. Intrusion tolerance in distributed computing systems. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy* (Oakland, Calif., May 1991) pp. 110–121.
5. Dolev, D. and Malki, D. The Transis approach to high availability cluster communication. *Commun. ACM 39,* 4 (April 1996).
6. Franklin, M.K. and Reiter, M.K. The design and implementation of a secure auction service. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy* (Oakland, Calif., May 1995) pp. 2–14.
7. Franklin, M.K. and Reiter, M.K. Verifiable signature sharing. In L.C. Guillou and J. Quisquater, eds., *Advances in Cryptology—EUROCRYPT '95* (Lecture Notes in Computer Science 921) pp. 50–63. Springer-Verlag, 1995.
8. Franklin, M.K. and Yung, M. The varieties of secure distributed computation. In *Proceedings of Sequences II, Methods in Communications, Security and Computer Science* (June 1991) pp. 392–417.
9. Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game. In *Proceedings of the 19th ACM Symposium on the Theory of Computing* (May 1987) pp. 218–229.
10. Kent, S.T. Internet privacy-enhanced mail. *Commun. ACM, 36*, 8 (Aug. 1993), 48–60.
11. Lamport, L., Shostak, R., and Pease, M. The Byzantine generals problem. ACM *Transactions on Programming Languages and Systems, 4*, 3 (July 1982) 382–401.
12. Lewis, P.H. Auction of collectibles on the Internet. *The New York Times*, May 23, 1995.
13. Mills, D.L. Network Time Protocol (Version 2) specification and implementation. RFC 1119, Sept. 1989.
14. Moser, L.E., Melliar-Smith, P.M., Agarwal, D.A., Budhia, R.K., and Lingley-Papadopoulos, C.A. Totem: A fault-tolerant multicast group communication system. *Commun. ACM 39,* 4 (April 1996).
15. Neuman, B.C. and Ts'o, T. Kerberos: An authentication service for computer networks. *IEEE Commun. 32*, 9 (Sept. 1994).
16. Reiter, M.K. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (Fairfax, Va., Nov. 1994) pp. 68–80.
17. Reiter, M.K. The Rampart toolkit for building high-integrity services. In K.P. Birman, F. Mattern, and A. Schiper, eds., *Theory and Practice in Distributed Systems* (Lecture Notes in Computer Science 938), 99–110. Springer-Verlag, 1995.
18. Reiter, M.K. A secure group membership protocol. *IEEE Transactions on Software Engineering, 22*, 1 (Jan. 1996) 31–42.
19. Reiter, M.K., Franklin, M.K., Lacy, J.B., and Wright R.N. The Ω key management service. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security* (New Delhi, India, March 1996).
20. Schneider, F.B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys, 22*, 4 (Dec. 1990), 299–319.
21. Simmons, G.J. An introduction to shared secret and/or shared control schemes and their application. In G.J. Simmons, ed., *Contemporary Cryptology: The Science of Information Integrity*, 441–497. IEEE Press, 1992.

About the Author:

MICHAEL K. REITER is a principal investigator and member of the technical staff at AT&T Bell Laboratories. **Author's Present Address:** AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ; email: reiter@research.att.com