AFRL-IF-RS-TR-2004-321
Final Technical Report
November 2004

CONTAINMENT AND INTEGRITY FOR MOBILE CODE

Cornell University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J363

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

AFRL-IF-RS-TR-2004-321 has been reviewed and is approved for publication.

APPROVED: /s/

PATRICK M. HURLEY
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** NOVEMBER 2004	**3. REPORT TYPE AND DATES COVERED** Final Jun 99 – Mar 04

4. TITLE AND SUBTITLE
CONTAINMENT AND INTEGRITY FOR MOBILE CODE

5. FUNDING NUMBERS
C - F30602-99-1-0533
PE - 63760E
PR - H541
TA - 10
WU - 01

6. AUTHOR(S)
Fred B. Schneider and
Andrew C. Myers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cornell University
120 Day Hall
Ithaca New York 14853

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency	AFRL/IFGA
3701 North Fairfax Drive	525 Brooks Road
Arlington Virginia 22203-1714	Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-321

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Patrick M. Hurley/IFGA/(315) 330-3624/ Patrick.Hurley@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

Significant progress was made on general approaches for security policy enforcement and for building systems that are both fault-tolerant and secure.
The work on policy enforcement is based on program analysis and program rewriting, the foundations of language-based security. Authorization, confidentiality, and integrity policies were addressed in a rich model that even admits mutual distrust among principals. Also, a formal characterization was developed for what policies can be enforced by various mechanisms.
With regard to composing security and fault-tolerance, proactive threshold cryptographic protocols were developed and studied. Various prototype systems were built to evaluate the practicality of these protocols and the approach. The protocols make extremely weak assumptions about the system in which they are deployed.

14. SUBJECT TERMS
Security Policy Enforcement, Language Based Security, Trustworthy Systems, Distributed Trust, Proactive Secret Sharing, Threshold Cryptography

15. NUMBER OF PAGES
18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Summary of Accomplishments

Under the auspices of this DARPA funding, we were successful in developing language-based techniques for enforcing security policies and in developing protocols to support the construction of systems that are both fault-tolerant and secure. Specific accomplishments include the following.

Inlined Reference Monitoring

- Prototype program rewriters to support the In-lined Reference Monitor (IRM) approach for security policy enforcement on Java Virtual Machine (JVM) language programs, Microsoft Intermediate Langauge (MSIL) programs, and Microsoft Baby-IL (BIL) programs [1, 2, 3].
- An improved implementation of the Java stack-inspection security model [3].
- A formal characterization of security policies that can be enforced by program rewriting [5].

Composing Security and Fault-tolerance

- Deployment of the Cornell On-line Certification Authority (COCA), a fault-tolerant and secure certification authority, as a proof of concept for a new approach to employing replication without sacrificing confidentiality [35].
- Development of a prototype publish/subscribe service to better understand the generality of the COCA architecture.
- Design, implementation, and analysis of (i) APSS (asynchronous proactive secret sharing) [36], the first proactive secret sharing protocol that does not require assumptions about execution timings or message delivery delays and (ii) a distributed blinding protocol [34] for that same asynchronous setting.

Information Flow Security

- A publicly available implementation of the Jif (Java information flow) compiler that controls flow of sensitive information in Java programs [12, 14].

- Polyglot, an extensible compiler framework for building Java language extensions [8, 13].
- The development of a new approach to building secure distributed systems, secure program partitioning, along with an implementation of this approach [11, 30, 31].
- Protocols for automatically replicating code and data in a distributed system in order to satisfy both confidentiality and integrity properties [32].
- The first program analysis that provably enforces strong information security properties (noninterference) for an expressive language with first-class functions and complex control flow [26, 28].
- A new language-based characterization of security for concurrent system with type systems for security policy enforcement [29].
- Identification of a new end-to-end security property, *robustness*, that characterizes systems that release sensitive information without permitting information laundering [27]. An extension to this property, *qualified robustness*, gives untrusted code limited control over information release. A simple program analysis was also developed that provably enforces robustness and qualified robustness.

Stateless, Reliable Communication

- Design and implementation of a stateless, efficient, reliable network communications protocol that is TCP-compatible and TCP-friendly, yet supports instant failover, per-packet service replication, and intrinsic resistance to denial of service.

The list of 36 publications supported by this funding appears as the final section of this report. No patents were filed.

2 Detailed Description of Technical Progress

2.1 Inlined Reference Monitoring

An alternative to placing the reference monitor and the target system to be monitored in separate address spaces is to modify the target system code, effectively merging the reference monitor in-line. But a reference monitor can only be merged into a target application provided the target can be prevented from circumventing the merged code.

Specifying such an *in-lined reference monitor* (IRM) involves defining

- *security events*, the policy-relevant operations that must be mediated by the reference monitor;
- *security state*, information stored about earlier security events that is used to determine which security events can be allowed to proceed; and
- *security updates*, program fragments that are executed in response to security events and that update the security state, signal security violations, and/or take other remedial action (e.g., block execution).

A load-time, trusted *IRM rewriter* merges checking code into the application itself, using program analysis and program rewriting to protect the integrity of those checks. The IRM rewriter thus produces a *secured application*, which is guaranteed not to take steps violating the security policy being enforced.

Over the past several years, we have developed a series of IRM rewriters. The ultimate goal is to enable a full-scale commercial deployment of the technology. And while we have not yet seen that deployment, our work funded under this grant has attracted interest from SUN Microsystems (for Java) and Microsoft Corporation (for .NET).

SUN Microsystem's interest stems from our success with the PSLang/PoET second-generation IRM rewriter [3] for JVML in reproducing prior implementations of the Java stack inspection policy (with comparable performance) and then in supporting a new implementation of the policy (with superior performance). Our new implementation of stack-inspection works by carefully allocating work, so that frequently executed JVM instructions bear relatively less of the burden associated with enforcement. The implementation exhibits performance that is competitive with the JVM-resident stack inspection implementation included in the commercial Java distribution.

We then started to investigate issues associated with integrating an IRM rewriter into an operating system. We chose Microsoft Windows as our host and commenced building an IRM rewriter for Microsoft's .NET architecture. That prototype implemented an aspect-oriented programming metaphor for Microsoft's CLR assembly language. An aspect-oriented program comprises *aspects*, each of which consists of a *point-cut* and some *advice*. The point-cut is a predicate that specifies where to do rewriting in target code, and the advice specifies how to do the rewriting. Designing a point-cut language that provides complete visibility at a high-level into an assembly language is an interesting challenge, and we have not arrived at a satisfactory solution—the essence of working at a high-level is to obscure implementation details, yet knowledge of those details is frequently important in defining the enforcement mechanism.

Finally, we started development of a type system to specify security policies for BIL (Baby Intermediate Language), a realistic subset of Microsoft's MSIL (Microsoft Intermediate Language). A rich class of security policies could now be specified as types; the type checker ensures that a program satisfies the policy, augmenting a non-compliant program with corrective actions if necessary. When this work is completed, the result will be a compile-time way to enforce what an in-lined reference monitor can handle plus some additional policies (that are in the class of policies that require program rewriting).

On the theoretical side, we developed a characterization of how powerful various language-based security mechanisms are for enforcing security policies [5]. Our work on security automata [15] offered a formal characterization for what policies can be enforced by monitoring and halting a target system's execution that is about to violate security requirements. We have now extended that work to give in [5] formal characterizations for three other classes of enforcement mechanisms:

- mechanisms that analyze the target system before it is executed,
- edit-automata enforcement mechanisms, and
- mechanisms that modify a target system before execution.

The last class corresponds to IRMs in their most general form and includes our prototype .NET in-lined reference monitor realization.

We can show that the class of policies precisely supported by static analysis is supported by both reference monitors and program rewriting; we also found that introducing a computability requirement on reference monitors was necessary but not sufficient for precise characterization of the class of policies actually realizable by reference monitors. And we identified a new condition, which we called *punctuality*, that seems essential for defining more accurate upper bounds on the power of reference monitors.

2.2 Composing Security and Fault-tolerance

Fault-tolerance and attack-tolerance are crucial for implementing trustworthy services, yet composing fault-tolerance and attack-tolerance can be a real challenge. For one thing, separation of concerns does not apply, because approaches to implementing fault-tolerance can reduce a system's attack-tolerance. An example is n-fold replication of a secret s, which adds fault-tolerance and improves the availability of s but does so by increasing from 1 to n the number of sites that must resist attacks to preserve the confidentiality of s.

To investigate these issues, we embarked on a project to build a fault-tolerant and attack-tolerant on-line certification authority. The result, COCA (Cornell On-line Certification Authority) [35], was successfully deployed both in a local area network and in the Internet.

A few basic elements were combined in novel ways to realize COCA:

- Replication based on a Byzantine quorum system was employed to achieve availability.
- Proactive recovery with threshold cryptography was used for digitally signing certificates in a way that defends against mobile adversaries that attack, compromise, and control one replica for a limited period of time before moving on to another.
- No assumptions were made about execution speed and message delivery delays; channels are expected to exhibit only intermittent reliability; and with $3t + 1$ COCA servers up to t may be faulty or compromised.

The result was a system with inherent defenses to certain denial of service attacks because, by their very nature, weak assumptions are difficult for attackers to invalidate.

Prior to our work on COCA, few protocols had been developed for the *asynchronous* model of computation that characterized the set of weak assumptions we were willing to make. So considerable effort during this DARPA project was expended in developing a suitable family of protocols. Two notable successes were APSS (an asynchronous proactive secret sharing protocol) [36] and a new distributed blinding protocol [34]. Both protocols employed similar techniques for coping with the hostile environment that weak assumptions bring:

(i) Solving agreement problems by computing multiple results—one for each outcome of the agreement—and labeling each computed result according to agreement outcomes on which it is contingent.

(ii) Tolerating intermittent channel outages by using repeated sends that are terminated by receiving a message signifying successful receipt of the original message.

(iii) Tolerating malevolent servers by including in messages information that allows a receiver to test whether a message it receives is consistent with the execution of a correct sender.

We have no illusions that the methodology employed to build COCA is universal. Some services must support operations that cannot be implemented in terms of COCA's quorum systems. The key question then is: What class of services can be supported using COCA's methods for combining Byzantine quorums and threshold cryptography. As a step towards answering this question, we prototyped two other services:

(1) a publish/subscribe channel for which availability, integrity, and confidentiality of published data must be preserved, and

(2) the CODEX (COrnell Data EXchange) distributed service for storage and dissemination of secrets [9].

2.3 Information Flow Security

The grant funding also resulted in several innovations for enforcing strong information security properties such as confidentiality and integrity, along with software that has been useful to researchers elsewhere.

Jif: Java information flow Under the auspices of this project, we implemented a compiler for the Jif programming language and made it publicly available for download [12]. Jif extends Java with protection of confidentiality via static information flow control. Advanced language features such as label polymorphism and run-time security labels are also supported by the Jif language; these features are important for implementing complex systems that interact with the outside world. Other researchers have used the Jif language to explore a variety of security topics, ranging from secure smartcards to privacy in web services.

Polyglot: an extensible compiler framework Polyglot is a Java class library that is easily extended through inheritance to create a compiler for a language that is a modification to Java [13]. Language extensions can be implemented without duplicating code from the framework itself [8, 30]. Polyglot is useful for implementing domain-specific languages, for exploring language design ideas, and for simplifying Java for pedagogical purposes. The framework has been used to implement both major and minor modifications to Java. Experience implementing several languages in this framework suggests that the cost of implementing language extensions scales well with the degree to which the language extends Java. Polyglot has been used not only to implement the Jif compiler but also has been used by several other research projects to build their own domain-specific language extensions.

Secure program partitioning One major research thrust was to build on the Jif compiler to explore secure program partitioning, an innovative way of building distributed systems that are secure by construction [11, 30, 31]. We call this approach secure program partitioning. It is a way to ensure data confidentiality and integrity in a distributed system that contains untrusted hosts and mutually distrusting principals. In fact, systems are typically distributed precisely because the participants do not fully trust one another or their hosts. This problem is particularly relevant to information systems comprising mutually distrusting organizations, such as the dynamic coalitions that arise in military settings.

In this new approach, secure programs are automatically partitioned by the Jif/split compiler back end into communicating subprograms that run on the various available hosts. Partitioning thus automatically extracts a secure communications protocol. The program is partitioned so that if any host is subverted, only a principal that has explicitly stated trust in the host needs to fear a violation of confidentiality. For a given principal p, the partitioned program is robust [27] against attacks on hosts not trusted by p.

The Jif/split compiler was implemented as an extension to the Jif compiler. This implementation includes not only the compiler that checks and partitions programs, but also a distributed run-time system that securely executes the resulting programs while guarding against subverted or malicious hosts. Performance of the system is quite reasonable, despite fine-grained program partitioning.

We also extended this work to automatically employ replication (in addition to partitioning) in order to satisfy confidentiality and integrity properties [32]. A key problem that arises in many distributed systems is how to obtain sufficient assurance of integrity. Replication is a useful technique because it adds additional integrity assurance when multiple hosts compute the same result and agree with each other. The current Jif/split compiler now replicates information and computation as necessary to achieve this assurance. New protocols have been developed to permit the secure transfer of control between one group of replicas and another. The fundamental security guarantee is unchanged, however: a principal's security is threatened only if a host that principal trusts is compromised. The technique was been shown effective for developing various secure auction protocols and a distributed game.

Information Flow Properties We made three advances in developing the theory of secure systems with respect to confidentiality and integrity security properties. First, we developed a security type system for provable enforcement of noninterference in low-level languages [26]. Our type system is an important step towards enabling security verification that low-level (machine) code satisfies strong confidentiality and integrity properties. Ours is the first work to show that any similarly expressive programming language can enforce these properties. This was, in addition, the first demonstration that noninterference could be enforced automatically for language with both first-class functions and a writable memory.

Another area of progress has been information flow security for concurrent systems. Concurrency makes both the theory and practice of security difficult. First, it introduces covert information channels that commonly used "possibilistic" theories of information flow simply ignore; these channels can be exploited by a malicious attacker. Second, existing static analyses for checking the security of concurrent systems are too restrictive; important secure concurrent programming idioms are rejected by these analyses. We have developed a new language-based model of information security for concurrent systems and shown that a statically typed programming language can be used to enforce this security model [29]. Encouragingly, static security checking for this language avoids some of the restrictiveness of previous techniques.

A third advance was on the security that can be provided in the presence of declassification and other downgrading mechanisms. In earlier work we identified a theoretical property called *robustness* that such a system could be expected to provide. The insight behind robustness is that an attacker or user of the system should not be able to exploit downgrading mechanisms to extract more information from the system than was intended by the programmer. We discovered how to compactly express this robustness condition in a language-based setting [27]. We earlier conjectured that the *robust declassification rule* used by the Jif/split compiler would enforce some kind of robustness property; we validated this conjecture by proving that the type system does enforce our language-based formalization of robustness. This proof increases the assurance provided by our secure program partitioning technique.

Certain systems that are not robust would still be considered secure, because untrusted code needs to be granted strictly delimited ability to cause the release of secret information. For example, in a distributed game in which no player trusts the other players, other players can make moves in the game that should cause information release. To capture the security of such systems, we introduced a generalization of robustness called *qualified robustness*. We also proved (in a core language setting) that the typing rules that we introduced in the Jif/split compiler do soundly enforce this new security property.

2.4 Stateless, Reliable Communication

Denial of service (DOS) attacks often rely on exhausting some server resource, which may be server bandwidth but also might be some other resource (such as memory or network sockets). SYN packet flooding is an example of the latter—TCP requires servers to expend kernel and application resources for maintaining per-connection state. We investigated a general solution to this problem: a TCP-like network protocol that imposes no per-connection state on the server side kernel and, therefore, exhibits an inherent resistance to

DOS attacks [25]. In our architecture, many server applications (such as web servers) can also avoid storing per-connection state. The key idea is that the kernel can piggyback connection state onto the packets that it sends to the client. Thus the client becomes responsible for maintaining connection (and server application) state, reducing the load on the server. The connection state is protected by a message authentication code (MAC) to prevent clients from tampering with it. We have implemented this protocol and shown that performance is close to that of standard TCP and that the protocol is TCP-friendly. We have also implemented instant failover and load balancing among servers, with no client-detectable disruption even on open connections. This is possible because servers store no connection state.

3 Publications Produced

(1) U. Erlingsson and F. B. Schneider. SASI enforcement of security policies: A retrospective. *Proceedings New Security Paradigms Workshop* (Ontario, Canada, Sept. 1999).

(2) Ulfar Erlingsson and Fred B. Schneider. SASI enforcement of security polices: A retrospective. *DARPA Information and Survivability Conference and Exposition (DISCEX'00)* (Hilton Head, South Carolina, January 2000), IEEE Computer Society, Los Alamitos, California, 287–295.

(3) Ulfar Erlingsson and Fred B. Schneider. IRM enforcement of Java stack inspection. *Proceedings 2000 IEEE Symposium on Security and Privacy* (Oakland, California, May 2000), IEEE Computer Society, Los Alamitos, California, 246–255.

(4) David Gries and Fred B. Schneider. Formalizations of substitutions of equals for equals. *Millennial Perspectives in Computer Science, Proceedings of the 1999 Oxford-Microsoft Symposium in honour of Professor Sir Antony Hoare*, (Davies, Roscoe, and Woodcock eds.) Palgrave Publishers, Hampshire, England. November 2000, 119–132

(5) Kevin W. Hamlen, Fred B. Schneider, and Greg Morrisett. Computability classes for enforcement mechanisms. To appear, *ACM Transactions on Programming Languages and Systems.*

(6) D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov. NAP: Practical fault-tolerance for itinerant computations. *Proc. 19th IEEE International Conference on Distributed Computing Systems* (June 1999, Austin, Texas), IEEE, 180–189.

(7) Dag Johansen, Robbert van Renesse, and Fred B. Schneider. WAIF: Web of asynchronous information filters. *Future Directions in Distributed Computing*, Lecture Notes in Computer Science, Volume 2585 (Schiper, Shvartsman, Weatherspoon, and Zhao, eds.) Springer-Verlag, 2003, 81–86.

(8) Jed Liu and Andrew C. Myers. JMatch: Iterable abstract pattern matching for Java. *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages* (New Orleans, LA, January 2003), 110–127.

(9) Michael A. Marsh and Fred B. Schneider. CODEX: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing 1*, No. 1 (January-March 2003), 34–47.

(10) Yaron Minsky and Fred B. Schneider. Tolerating malicious gossip. *Distributed Computing* 16, 1 (Feb 2003), 49–68.

(11) Andrew C. Myers. Security-typed languages and distributed computation. *Static Analysis, Proceedings 8th International Symposium SAS 2001* (Paris, France, July 2001), Lecture Notes in Computer Science Volume 2126, Springer-Verlag, Heidelberg, 2001, 437.

(12) Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4), October 2000, 410–442.

(13) Nathaniel Nystrom, Michael R. Clarkson, and Andrew C. Myers. Polyglot: An extensible compiler framework for Java compiler construction. *Proceedings of the 12th International Conference on Compiler Construction* (Warsaw, Poland, April 2003), 138–152.

(14) Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* Special issue on Design and Analysis Techniques for Security Assurance, 21(1) (January 2003), 5–19.

(15) Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security* 3, 1 (February 2000), 30–50.

(16) Fred B. Schneider. Open source in security: Visiting the bizarre. *Proceedings 2000 IEEE Symposium on Security and Privacy* (Oakland, California, May 2000), IEEE Computer Society, Los Alamitos, California, 126–127.

(17) Interview with Fred B. Schneider. *Distributed Systems Online.* http://www.computer.org/channels/ds.

(18) Fred B. Schneider. Editorial: Time for change. *Distributed Computing* Vol. 13, No. 4 (November 2000), 187.

(19) Fred B. Schneider. Language-based security: What's needed and why. *Static Analysis, Proceedings 8th International Symposium SAS 2001* (Paris, France, July 2001), Lecture Notes in Computer Science Volume 2126, Springer-Verlag, Heidelberg, 2001, 374.

(20) Fred B. Schneider. Least privilege and more. *Computer Systems: Papers for Roger Needham*, Andrew Herbert and Karen Sparck Jones, eds. Microsoft Research, 2003, 209–213.

(21) Fred B. Schneider. Least Privilege and more. *IEEE Security and Privacy*, Volume 1, Number 3 (September/October 2003), 55–59.

(22) Fred B. Schneider and Mike Rodd, eds. International review of UK research in Computer Science. EPSRC, BCS, and IEE Technical Report. October, 2001.

(23) F.B. Schneider, S. Bellovin, and A. Inouye. Building trustworthy systems: Lessons from the PTN and Internet. *IEEE Internet Computing*, 3, 5 (November-December 1999), 64–72.

(24) Fred B. Schneider, Greg Morrisett, and Robert Harper. A language-based approach to security. *Informatics: 10 Years Back, 10 Years Ahead.* Lecture Notes in Computer Science, Vol. 2000 (Reinhard Wilhelm, editor), Springer Verlag, Heidelberg, 2000, 86–101.

(25) Alan Shieh, E. Gn Sirer, Andrew C. Myers. Trickles: A stateless network stack for scalability, resilience, and flexibility. Submitted for publication.

(26) Steve Zdancewic and Andrew C. Myers. Secure information flow and CPS. *Proceedings of the 10th European Symposium on Programming* (Genova, Italy, April 2001), 46–61.

(27) Steve Zdancewic and Andrew C. Myers. Robust declassification. *Proceedings of the 14th IEEE Computer Security Foundations Workshop* (Cape Breton, Nova Scotia, Canada), June 2001.

(28) Steve Zdancewic and Andrew C. Myers. Secure information flow and linear continuations. *Higher-Order and Symbolic Computation 15*, No. 2–3 (Sept. 2002), 209–235.

(29) Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. *Proc. 16th IEEE Computer Security Foundations Workshop*, (Pacific Grove, California, June 2003), 29–43.

(30) Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Untrusted hosts and confidentiality: Secure program partitioning. *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (Banff, Canada, October 2001), 1–14. Award paper.

(31) Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *ACM Transactions on Computing Systems 20*, No. 3 (August 2002), 283–328.

(32) Lantian Zheng, Stephen Chong, Andrew C. Myers, Steve Zdancewic. Using replication and partitioning to build secure distributed systems. *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, (Oakland, California, May 2003), 236–250.

(33) L. Zhou, and Z. Haas. Securing ad hoc networks. *IEEE Networks* 13, 6 (November-December 1999), 24–30.

(34) L. Zhou, Michael A. Marsh, Fred B. Schneider, and Anna Redz. Distributed blinding for distributed Elgamel re-encryption. Submitted for publication.

(35) L. Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems 20*, No. 4 (November 2002), 329–368.

(36) L. Zhou, Fred B. Schneider, and Robbert van Renesse. APSS: Proactive secret sharing in asynchronous systems. Submitted for publication.