On Adversary Models and Compositional Security

A unified view of a wide range of adversary classes and composition principles for reasoning about security properties of systems are cornerstones of a science of security. They provide a systematic basis for security analysis by explaining and predicting attacks on systems.

Anupam Datta, Jason Franklin, Deepak Garg, Limin Jia, and Dilsun Kaynar
Carnegie Mellon University

In this article, we report a representative result in the science of security. To explain what we mean by a "science," we can draw an analogy with physics. A physical theory consists of a model of the physical universe. The model should be general—that is, it should encompass a large class of physical phenomena. It should also support analyses that identify relationships among physical concepts, which researchers can then use to explain observed behavior in the physical universe and predict behavior that they haven't yet observed. For example, Albert Einstein's general theory of relativity presents a model of gravitation—a set of equations that describe how spacetime is curved by matter and energy (a relationship among physical concepts). It explains observed phenomena, such as the bending of light near the sun, and predicts the existence of black holes, regions of spacetime where the gravitational attraction is so strong that even light can't escape. The theory is general in that its predictions apply to a very large class of phenomena ranging from motion of bodies (apples, stars, planets) in free fall to the propagation of light.

A science of security should include theories for the security universe that have similar characteristics. The security universe includes a large class of computer systems (Web browsers, hypervisors, virtual machine monitors, operating systems, trusted computing systems, and network protocols, to name a few) that are intended to provide subtle security properties in the presence of adversaries who actively interfere with a system's execution. A security theory should therefore include a model for systems, adversaries, and properties that supports analyses that identify relationships among classes of systems, adversaries, and properties. These relationships should in turn help explain observed phenomena (why specific attacks work against specific systems) and predict phenomena (how well a system will hold up as adversaries launch new attacks). A theory is general if it applies to large classes of systems, adversaries, and properties.

In this article, we present the outline of a theory of compositional security that addresses a recognized scientific challenge.[1] Contemporary systems evolve from smaller components, but even if each component is secure in isolation, the composed system might not achieve the desired security property because an adversary could still exploit complex interactions between components to compromise security. A theory of compositional security should identify relationships among systems, adversaries, and properties such that precisely defined composition operations over systems and adversaries preserve security properties. Such a theory would thus enable scalable analysis of large, complex systems by constructing their security proofs from separately constructed proofs of properties of the simpler components from which they're built. In addition, if a component is used to build multiple systems, the proof of its security property could be reused in the proofs for all systems constructed.

Although researchers have made progress in understanding secure composition in specific settings,

 1540-7993/11/$26.00 © 2011 IEEE

such as information flow control for noninterference-style properties[2] and cryptographic protocols,[3–5] a systematic understanding of the general problem of secure composition hasn't emerged yet. Our theory builds on and generalizes prior work on a compositional theory for the domain of cryptographic protocols[4] and is influenced by compositional reasoning principles for functional correctness of programs.[6,7] The security literature offers several alternative approaches to compositional security. In particular, one recent work applies the universal composability approach,[3,5] originally developed for cryptographic protocols, to systems.[8] We refer the interested reader to a technical paper we've written for comparison with additional related work.[9]

Modeling Both Systems and Adversaries

We model a system as a set of concurrently running—and possibly interacting—threads of programs that access a set of resources only through stipulated interfaces. Figure 1a illustrates our model's key components. In the figure, R_i and I_i represent resources and interfaces, respectively. Trusted components, T_i, combine interface calls in known ways. Adversarial (untrusted) components, A_i, on the other hand, can combine calls to interfaces they access arbitrarily. In general, the set of interfaces I_{Ai} available to adversaries is a subset of the set of interfaces I_{Ti} available to the trusted system components.

This model is general: it captures a wide range of real systems and associated adversary models. For example, Figure 1b shows a model of Web mashups obtained by instantiating the elements of Figure 1a. Resources include the mashup's document object model (DOM), communication channels between frames, and the network. Each frame in the mashup corresponds to one thread of the system. An adversary is a set of malicious frames. Interfaces I_D for accessing the DOM include functions for reading and writing DOM elements; interfaces for interframe communication include the postmessage method; and interfaces for the network include methods for obtaining data and code over the network. Note that adversarial frames are limited in their behavior by these interfaces—if all network interfaces restrict communication to servers in the same domain as the originating frame (the so-called same-origin policy), then an adversarial frame can't contact a server from a different domain regardless of its program.

Another example obtained by instantiating Figure 1a is that of a file system (Figure 1c). Here, the resources are files, the data structure holding the access permission matrix, and possibly the network. As usual, the model assumes that the administrator (such as the superuser in Unix-based systems) is trusted and that other users might be adversarial. Again, adversaries can't break the interface abstraction: if no interface allows Mallory to read secret.txt, Mallory's program can't read secret.txt regardless of the instructions it executes.

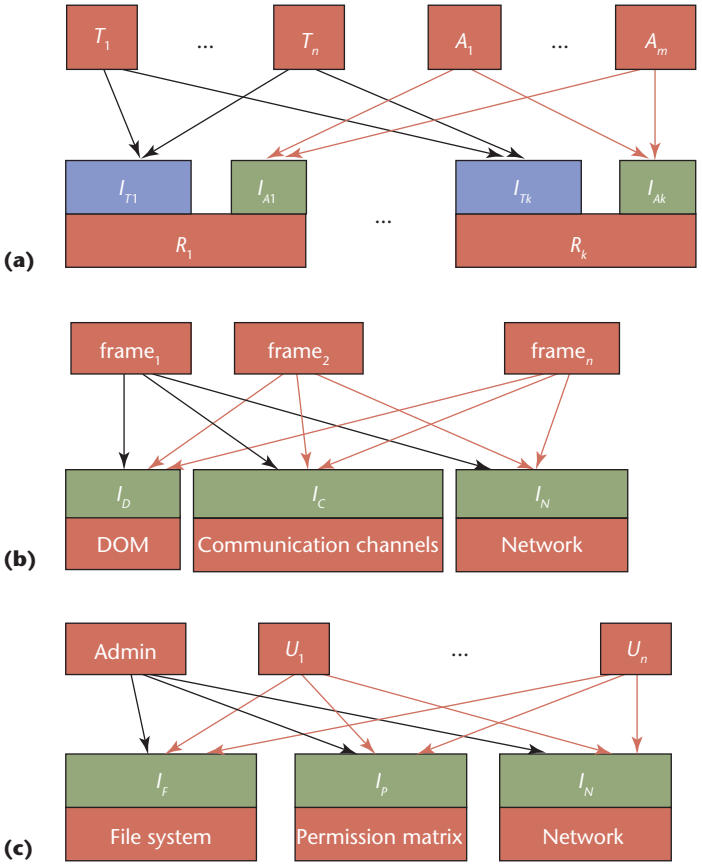

Figure 1. (a) Abstract interface view of a system and its instantiations to (b) a Web mashup and (c) a file system. Trusted components combine interface calls in known ways. Adversaries can combine interface calls in arbitrary ways but can't break the interface abstraction.

Yet another instance of our abstract model of systems is a network security protocol in which we view the network as the sole shared resource of interest and in which interfaces include message send and receive functions, encryption, decryption, and nonce generation. Trusted threads follow their parts of the protocol, whereas adversarial threads combine interface calls any way they choose. Yet again, adversaries are confined by the interfaces available to them—they can intercept and send messages, but they can't decrypt messages that are encrypted using keys that they don't know.

At a technical level, interfaces are modeled as recursive functions in an expressive programming language. Trusted components and adversaries are also represented using programs in the same programming

language. Typically, we assume that the programs for the trusted components (or their properties) are known. On the other hand, an adversary is modeled by considering all possible programs that can be constructed by combining calls to the interfaces to which the adversary has access.

Modeling Security Properties

In defining models for security properties, a useful abstraction is that of a trace, which is a possible sequence of events obtained via system execution. Our model focuses on trace properties, specifically, safety properties. Informally, safety properties state that "nothing bad ever happens on the trace." Formally, a trace violates a safety property if and only if the trace has a finite prefix on which the property is violated. In contrast, liveness properties state that "something good eventually happens on a trace."

We focus on safety properties for the following reasons:

- safety properties are general enough to either express or approximate most security properties of interest, including authorization, integrity, secrecy, and information flow properties;
- interfaces can reliably guarantee only safety properties—although a file system interface might guarantee that Mallory never reads secret.txt (a safety property), no file system interface can guarantee that secret.txt will eventually be read (a liveness property);[10] and
- safety properties are possibly amenable to compositional reasoning, but common compositional reasoning principles such as rely-guarantee reasoning don't apply to liveness properties.[7]

We represent security properties as formulas in a first-order temporal logic, following prior work on modeling functional correctness properties.

Compositional Security

Two compositional reasoning principles capture relationships among systems, adversaries, and properties in our model. These relationships are general; they explain why certain attacks work against specific systems and predict when specific systems will preserve their security properties even as adversaries come up with new attacks.

Composition Principle 1

If two system components T_1 and T_2 satisfy properties φ_1 and φ_2 in isolation, respectively, does their simultaneous execution $T_1 \| T_2$ satisfy $\varphi_1 \wedge \varphi_2$? Equivalently, assuming we've proven that a trusted component T_1 satisfies property φ_1, can we prove that the simultaneous execution of T_1 and another component T_2 still satisfies φ_1? It's easy to see that the latter isn't true for all properties φ_1. For instance, let component T_1 contain two concurrent threads, A_1 and B_1, executing the simple protocol in which A_1 sends a message to B_1, and B_1 sends back an acknowledgment. Then let φ_{AB1} be the property: if A_1 receives an acknowledgment from B_1, then B_1 received a message earlier. Clearly, T_1 in isolation satisfies φ_{AB1}. However, in a system in which message senders can spoof their identities, simultaneous execution of T_1 with an adversarial thread T_2 that simply sends an acknowledgment to A_1, spoofing its origin to be B_1, no longer satisfies the property φ_{AB1} because A_1 might receive an acknowledgment from T_2 without B_1 having received a message.

Even though not all security properties are compositional, certain properties—namely, those that mention only the actions (or activities) of a single thread—are compositional. We use the term *local* for such properties. For instance, the property φ_{B1}—which says that if B_1 sends an acknowledgement, then it must have received a payload earlier—is local. The fact that local properties compose is captured in the following rule for local properties φ_1 and φ_2, where $\vdash T : \varphi$ means that thread T satisfies property φ:

$$\frac{\vdash T_1 : \varphi_1 \quad \vdash T_2 : \varphi_2}{\vdash T_1 \| T_2 : \varphi_1 \wedge \varphi_2}.$$

Although local properties compose, most security properties of interest, such as φ_{AB1}, aren't local. How, then, might we develop compositional proofs for security properties in general? The critical observation that lets us proceed is that because a security property is a consequence of actions of individual threads, we can factor the security property's proof into proofs of local properties, followed by reasoning that combines these local properties. This combination step, called *global reasoning* in the sequel, often relies on domain-specific assumptions about the system—that is, assumptions that apply to all system components. For instance, in network protocol analysis, the assumption that a message can't be decrypted without proper keys is domain-specific. Such assumptions can either be axiomatic or established through an analysis of interfaces, as described later. Continuing our earlier example, suppose we make the domain-specific assumption that sender identities can't be spoofed. We can prove $\vdash T_1 : \varphi_{AB1}$ as follows. First, we establish that the local property φ_{B1} holds. Then, we complete the proof by global reasoning: if A_1 receives a message purportedly from B_1, then because of the domain-specific assumption, B_1 must have sent the message, and because of φ_{B1}, we can conclude that

B_1 must have received a payload earlier. Interestingly, this proof remains virtually unchanged when we add the malicious thread T_2 because $\vdash T_1 \| T_2 : \varphi_{B1}$ follows from the composition rule presented earlier and $\vdash T_1 : \varphi_{B1}$ (choose $\varphi_1 = \varphi_{B1}$ and $\varphi_2 =$ true in the above rule), whereas the step of global reasoning is unchanged. Thus, by factoring proofs into proofs of local properties followed by reasoning from domain-specific assumptions about the system, we obtain fully compositional proofs of security.

Our interface-based model is useful in justifying domain-specific assumptions made in the global reasoning step described earlier. Because we assume that all threads in a system are confined to a stipulated set of interfaces, we can treat any invariant preserved by all interfaces in the set as a domain-specific assumption. Formally, if we can prove that a component T (possibly composed of several other components) satisfies property φ under the domain-specific assumption φ_A, written $\varphi_A \vdash T : \varphi$, and we can prove that all allowed interfaces I_A preserve φ_A, written $\vdash_{I_A} \varphi_A$, then T satisfies φ. This is captured in the following rule:

$$\frac{\vdash_{I_A \varphi_A} \quad \varphi_A \vdash T : \varphi}{\vdash T : \varphi}.$$

Therefore, from a composition perspective, the following style of proofs is beneficial:

- *local reasoning*—prove local properties of known threads by analyzing their programs;
- *interface analysis*—prove invariants by analyzing the interfaces available to threads; and
- *global reasoning*—combine local reasoning and interface analysis by logical deduction to complete the proof.

Any such proof is compositional—it's correct regardless of what other components (possibly adversarial) exist, provided that the other components are confined to the interfaces considered in the interface analysis step.

Let's review a published analysis of the widely deployed Trusted Computing technology using our method,[11] and the consequent discovery of a real incompatibility between an existing standard protocol for attesting the software stack's integrity to a remote party and a newly added hardware instruction. Machines with Trusted Computing abilities include a special, tamper-proof hardware called the Trusted Platform Module (TPM), which contains protected append-only registers to store measurements, or hashes, of programs loaded into memory and a dedicated coprocessor to sign the contents of the registers with a unique hardware-protected key. The protocol in question, called Static Root of Trust Measurement (SRTM), uses this hardware to establish the software stack's integrity on a machine to a remote third party. The protocol works by requiring each program to store in the protected registers the hash of any program it loads. For example, the hash of the first program loaded into memory, usually the boot loader, is stored in the protected registers by the booting firmware, usually the Basic Input/Output System (BIOS). The stack's integrity following this protocol can be proved to a third party by asking the coprocessor to sign the contents of the protected registers with the hardware-protected key and then sending the third party the signed hashes. The third party can compare these hashes to known ones, thus validating the stack's integrity.

Note that the SRTM protocol is correct only if software that hasn't already been measured can't append to the protected registers. Indeed, this invariant was true in the hardware the initial Trusted Computing standard prescribed, so this protocol was secure at that point. However, a new instruction, called `latelaunch`, added to the standard in a later extension allows an unmeasured program to be started with full access to the TPM. This violates the necessary invariant and results in an actual attack on the SRTM protocol: a program invoked with `latelaunch` can add hashes of arbitrary programs to the protected registers without actually loading them. Because the program isn't measured, the remote third party obtaining the signed measurements will never detect its presence. An analysis of the protocol using our method as outlined here easily discovered this incompatibility between the SRTM protocol and the `latelaunch` instruction. In the analysis, the TPM instruction set (which included `latelaunch`) was modeled as an available interface to programs. The necessary invariant can be established for all interfaces except `latelaunch`, thus leading to failure of a proof of correctness with `latelaunch` and to discovery of the actual attack.

Composition Principle 2

Although the structure of proofs presented above is very general, it doesn't suffice for proving those inductive security properties that hold at a point in time if and only if they've held at all prior points in time. Consider the following two examples:

A failure to complete an expected proof step might help explain why a specific system doesn't satisfy a security property.

- A file system whose access control mechanism includes a special permission "admin" that lets a user modify permissions for other users. Suppose that, initially, only Alice and Bob have admin permission, and that the programs Alice and Bob run never provide the admin permission to anyone. The property of interest is *no principal besides Alice or Bob ever has the admin permission*.
- An operating system kernel mechanism that stores page tables in a protected area of memory. Initially, the page tables map all virtual addresses to physical addresses outside the protected area. The property of interest is *the page tables never map any virtual address in the protected area*.

In both cases, there are data structures (access control list in the first case and page tables in the second) that protect themselves from modification. In both cases, the proof that the respective property holds at a particular point in time relies on the property having been true at all points in the past. We can prove the first example as follows: if its property doesn't hold at time t, then someone other than Alice or Bob must have added the admin permission for someone other than Alice or Bob before time t. So the former principal also had admin permission at that earlier time, hence the property didn't hold then either. A similar argument applies to the second example. Formally, these proofs proceed by induction over traces. Can we structure these inductive proofs so that they're compositional—that is, they're valid regardless of what other components execute simultaneously?

Fortunately, we can make such inductive proofs compositional by combining ideas from the previous section with a well-understood style of proofs called rely-guarantee reasoning.[6,7] Suppose we want to prove that property φ holds at all times. First, we identify a set $S = \{T_1, \ldots, T_n\}$ of trusted threads relevant to the property and local properties $\psi_{T1}, \ldots, \psi_{Tn}$ of these threads, satisfying the following conditions:

1. If φ holds at all time points strictly before any given time point, then each of $\psi_{T1}, \ldots, \psi_{Tn}$ holds at the given time point
2. If φ doesn't hold at any time, then at least one of $\psi_{T1}, \ldots, \psi_{Tn}$ must have been violated strictly before that time.

The rely-guarantee principle states that under these conditions, if φ holds initially, then φ holds forever. We illustrate the technique by using it to prove the property φ of the example with Alice and Bob. We choose S to be the set of all threads in the system and ψ_T (for a thread T) to be the property that the thread T doesn't add the admin permission for anyone. Then, statement 1 above follows for Alice and Bob's threads because they don't give the admin permission to anyone and for other threads because to change the permissions, they must have the admin permission—that is, φ must be previously violated, which is ruled out by the assumption in statement 1. Statement 2 declares that if someone other than Alice or Bob has an admin permission, then some thread must have added that permission. This follows from a domain-specific assumption that permissions can't change on their own. Thus, the rely-guarantee principle implies that φ holds forever, as required. The important observation here is that this proof is completely compositional. Statement 1 proves local properties, which are compositional as discussed previously; statement 2 is trivially compositional because all components must adhere to it. Consequently, the proof is valid regardless of which threads execute in the system besides Alice and Bob's.

In general, any proof produced using this technique is compositional, and this reasoning method is compatible with reasoning about interfaces as well. In proving either statements 1 or 2, we can assume invariants that are satisfied by all interfaces available to programs in the system.

Another application of the rely-guarantee technique, different from verification of self-protecting data structures, is in proofs of key secrecy in network protocols. We explain one instance here—proving that the so-called authentication key (AKey) generated during the Kerberos V protocol becomes known only to three protocol participants:[9] the client authenticated by the key, the Kerberos Authentication Server (KAS) that generates the key, and the Ticket Granting Server (TGS) to whom the key authenticates the client. At the center of this proof is the property that whenever any of these three participants sends out the AKey over the (unprotected) network, it's encrypted with other secure keys. Proving this property requires induction because as part of the protocol, the client blindly forwards an incoming message to the TGS. Consequently, the fact that the client's outgoing message doesn't contain the unencrypted AKey relies on the fact that the incoming message doesn't contain the unencrypted AKey. The latter follows from the inductive hypothesis that any network adversary couldn't have had the unencrypted AKey to send it to the client.

Formally, the rely-guarantee framework is instantiated by choosing φ to be the property that any message sent out on the network doesn't contain the unencrypted AKey. ψ_T, for threads T of the client, the KAS, and the TGS, is the property that the respective threads don't send out the AKey unencrypted. Then, the proof of statement 2 is trivial, and statement 1 follows from an analysis of the client's programs, the

KAS, and the TGS. The first of these, as mentioned earlier, uses the assumption that φ holds at all points in the past. Note that the three programs are analyzed individually, even though the secrecy property relies on the interactions between them.

Predicting and Explaining Attacks

The composition principles described earlier are quite general, as demonstrated in the first principle (systematic combination of local reasoning, interface analysis, and global reasoning) by successfully proving authentication properties of network protocols[4] and integrity properties for trusted computing platforms.[11] The second principle (rely-guarantee reasoning) has been applied to compositionally prove self-protecting data structures' integrity properties[9] and network protocols' secrecy properties.[9,12]

Our model of interfaces can also help predict whether a system has a security property, given that it exposes certain interfaces to adversaries: if we assume only the invariant φ_A in proving a security property φ, then the system is secure provided that all interfaces it exposes maintain this invariant. This interface invariant thus abstractly characterizes a class of attacks that are ineffective against the system: any specific attack that doesn't break the invariant won't break the security property. Dually, if even one interface doesn't maintain this invariant, the system could potentially face attack. Of course, the attack might not be real because the assumption φ_A might not be essential to the proof (there could be another proof without the assumption), but such a failure could be used as a red flag during system design.

A failure to complete an expected proof step might help explain why a specific system doesn't satisfy a security property. For example, missing checks in interfaces could result in failure to prove invariants that are necessary for proving the security property. One concrete example of such vulnerability in a security hypervisor appears in a recent paper.[13] Another common source of attacks observed in practice arises from failure to consider certain interfaces available to adversaries. In this case, by omitting analysis of some interfaces, we may prove stronger invariants than ones that actually hold in the system and (incorrectly) use these invariants in proving security properties. We mentioned one such example earlier, in the context of TPMs; similarly, vulnerabilities have resulted from a failure to consider the direct memory access (DMA) write procedure as part of the interface available to adversaries.[14]

Future work in theory of compositional security can take several directions. First, automating the compositional reasoning principles we presented is an open problem. Rely-guarantee reasoning principles have already been automated for functional verification of realistic systems, and we expect that progress can be made by building on these prior results. Second, there's a strong need to develop and standardize domain-specific adversary models for system security. Although work exists on such models in some domains—network protocols and trusted computing platforms—we haven't yet arrived at a similar level of understanding in other important domains, such as the Web platform. Finally, it's important to extend the compositional reasoning principles presented here to support analysis of more refined models that consider, for example, features of implementation languages such as C. □

References

1. J.M. Wing, "A Call to Action: Look beyond the Horizon," *IEEE Security & Privacy*, vol. 1, no. 6, 2003, pp. 62–67.
2. D. McCullough, "A Hookup Theorem for Multilevel Security," *IEEE Trans. Software Eng.*, vol. 16, no. 6, 1990, pp. 563–568.
3. R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," *Proc. 42nd Ann. Symp. Foundations of Computer Science* (FOCS), IEEE CS Press, 2001, pp. 136–145.
4. A. Datta et al., "A Derivation System and Compositional Logic for Security Protocols," *J. Computer Security*, vol. 13, no. 3, 2005, pp. 423–482.
5. B. Pfitzmann and M. Waidner, "A Model for Asynchronous Reactive Systems and Its Application to Secure Message Transmission," *Proc. 21st IEEE Symp. Security and Privacy*, IEEE CS Press, 2001, pp. 184–200.
6. C.B. Jones, "Tentative Steps toward a Development Method for Interfering Programs," *ACM Trans. Programming Languages and Systems*, vol. 5, no. 4, 1983, pp. 596–619.
7. J. Misra and K.M. Chandy, "Proofs of Networks of Processes," *IEEE Trans. Software Eng.*, vol. 7, no. 4, 1981, pp. 417–426.
8. R. Canetti et al., "Composable Security Analysis of OS Services," Cryptology ePrint Archive, Report 2010/213, Int'l Assoc. for Cryptographic Research, 2010; http://eprint.iacr.org/2010/213.
9. D. Garg et al., "Compositional System Security in the Presence of Interface-Confined Adversaries," *Proc. 26th Conf. Mathematical Foundations of Programming Semantics* (MFPS), Elsevier, 2010, pp. 49–71.
10. J. Rushby, "Kernels for Safety?" *Safe and Secure Computing Systems*, T. Anderson, ed., Blackwell Scientific Publications, 1989, pp. 210–220.
11. A. Datta et al., "A Logic of Secure Systems and Its Application to Trusted Computing," *Proc. 30th IEEE*

Symp. Security and Privacy, IEEE CS Press, 2009, pp. 221–236.
12. A. Roy et al., "Secrecy Analysis in Protocol Composition Logic," *Formal Logical Methods for System Security and Correctness*, IOS Press, 2008.
13. J. Franklin et al., "Scalable Parametric Verification of Secure Systems: How to Verify Reference Monitors without Worrying about Data Structure Size," *Proc. 31st IEEE Symp. Security and Privacy*, IEEE CS Press, 2010, pp. 365–379.
14. L.J. Fraim, "SCOMP: A Solution to the Multilevel Security Problem," *Computer*, vol. 16, no. 7, 1983, pp. 26–34.

Anupam Datta *is a research faculty member at Carnegie Mellon University. His research interests are in trustworthy systems, privacy, and analysis of cryptographic protocols. He has served as general chair of the 2008 IEEE Computer Security Foundations Symposium, as program cochair of the 2008 Formal and Computational Cryptography Workshop, and on the program committees of many other computer security conferences. Datta has a PhD in computer science from Stanford University. Contact him at danupam@cmu.edu.*

Jason Franklin *is a PhD candidate in the Department of Computer Science at Carnegie Mellon University. His research focuses on the application of principled techniques to improve system and network security. Franklin has a BS in computer science and mathematics from the University of Wisconsin-Madison. He received the 2005 Usenix Security Best Paper Award, the 2009 SOSP Best Paper Award, a Department of Homeland Security Fellowship, and NSF Graduate Research Fellowship. Contact him at jfrankli@cs.cmu.edu.*

Deepak Garg *is a postdoctoral researcher at CyLab, Carnegie Mellon University. His research interests include formal logic, type-theory, access control, privacy, and analysis of system security properties. Garg has a PhD from Carnegie Mellon. Contact him at dg@cs.cmu.edu.*

Limin Jia *is a postdoctoral researcher at CyLab, Carnegie Mellon University. Her research interests include programming languages, language-based security, type systems, logic, and program verification. Jia has a PhD in computer science from Princeton University. Contact her at liminjia@cmu.edu.*

Dilsun Kaynar *is a postdoctoral researcher at CyLab, Carnegie Mellon University. Her research interests include formal modeling and verification of distributed system designs, and foundations of security and privacy. Kaynar has a PhD from the University of Edinburgh and did postdoctoral training at MIT, specializing in distributed computing theory. Contact her at dilsun@cs.cmu.edu.*