On Adversary Models and Compositional Security

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

Abstract

We present a representative development in the science of security that includes a generic model of computer systems, their security properties and adversaries who actively interfere with such systems. We describe logic-based methods to reason about security properties of a system as a composition of properties of its components, and several successful applications of the method in explaining and predicting attacks in a wide-variety of systems.

Keywords. Adversary models; compositional security; logic-based security; rely-guarantee reasoning

1 Introduction

This article reports on a representative result in the *science of security.* In order to explain what we mean by a "science", we draw an analogy with physics. A physical theory consists of a *model* of the physical universe. The model should be *general*, i.e., it should encompass a large class of physical phenomena. The model should also support analyses that identify *relationships* among physical concepts, which can then be used to *explain* observed behavior in the physical universe, and *predict* behavior that has not yet been observed. For example, 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 cannot 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* (e.g., web browsers, hypervisors, virtual machine monitors, operating systems, trusted computing systems, and network protocols) that are intended to provide subtle *security properties* (e.g., confidentiality and integrity of data) in the presence of *adversaries* who actively interfere with the execution of the system. A theory of security should therefore include a *model* for systems, adversaries, and properties. The model should support analyses that identify *relationships* among classes of systems, adversaries and properties. These relationships should in turn be used to *explain* observed phenomena (e.g., why specific attacks work against specific systems), and *predict* phenomena (e.g., how well a system will hold up as adversaries come up with new attacks). A theory is general if it applies to large classes of systems, adversaries and properties.

We now present the outline of a theory of *compositional security*[1], addressing a recognized scientific challenge (see, for example, Wing [15]). Contemporary systems are built up from smaller components. However, even if each component is secure in isolation, the composed system may not achieve the desired security property: an adversary may exploit complex interactions between components to compromise security. The goal of a theory of compositional security is to 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 are 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 the systems.

While there has been progress on understanding secure composition in specific settings, such as information flow control for non-interference-style properties [10] and cryptographic protocols [2, 4, 12], a systematic understanding of the general problem of secure composition has not emerged yet. In presenting our theory, we describe our model for general classes of systems, adversaries and security properties. We then present composition results (relationships) in this model. We also discuss how our theory explains a number of specific attacks found in the wild and how it can serve as the basis for predicting whether security properties of systems will be preserved as adversaries come up with new attacks.

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 [9, 11]. Several alternative approaches to compositional security have been considered in the literature. In particular, in recent work, the universal composability approach [2, 12], originally developed for cryptographic protocols, has also been applied to systems [3]. We refer the interested reader to our technical paper [8] for comparison with additional related work.

2 Modeling 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 1 illustrates the key components of our model. In the figure labels R_i and I_i represent the resources and interfaces respectively. Trusted components, labeled T_i, combine interface calls in known ways. Adversarial (untrusted) components, labeled A_i, on the other hand, can combine calls to interfaces they can access in arbitrary ways. In general, the set of

[1]The interested reader is referred to the associated technical paper [8] for additional details.

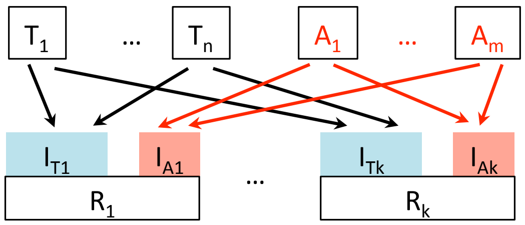

Figure 1: Interface view of a system

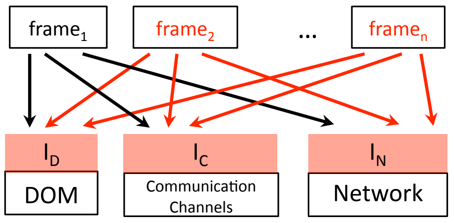

Figure 2: Interface view of a mashup

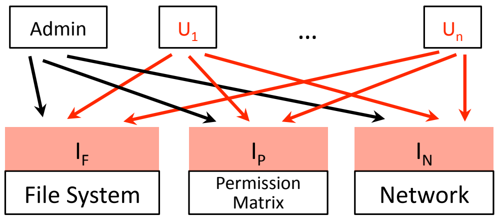

Figure 3: Interface view of a file system

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, a model of web mashups, obtained by instantiating the elements of Figure 1, is shown in Figure 2. Resources include the Document Object Model (DOM) of the mashup, communication channels between frames, and the network. Each frame in the mashup corresponds to one thread of the system and an adversary is a set of malicious frames. Interfaces for accessing the DOM I_D include functions for reading and writing elements in the DOM; interfaces for inter-frame communication include the postmessage method; 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, e.g., if all network interfaces restrict communication to servers in the same domain as the originating frame (the so-called same-origin policy), then regardless of its actual program, an adversarial frame cannot contact a server from a different domain.

Another example, also obtained by instantiating Figure 1, is that of a file system (Figure 3). 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 may be adversarial. Again, adversaries cannot break the abstraction of interfaces; if no interface allows Mallory to read `secret.txt`, Mallory's program cannot read `secret.txt` regardless of the instructions it executes.

Yet another instance of our abstract model of systems is a network security protocol where we view the network as the sole shared resource of interest and 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 in any way of their choosing. Yet again, adversaries are confined by the interfaces available to them — they can intercept and send messages, but cannot decrypt messages that are encrypted using keys that they do not 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. However, an adversary is modeled by considering all possible programs that can be constructed by combining calls to the interfaces to which the adversary is confined.

3 Modeling Security Properties

In defining models for security properties, a useful abstraction is that of a *trace* — the possible sequence of events that is obtained by executing a system. Our model focuses on properties of traces, specifically, *safety properties*. Informally, safety properties are negative properties that state "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. First, it is well-known that safety properties are *general* enough to either express or approximate most security properties of interest including authorization, integrity, secrecy, and information flow properties. Second, interfaces can reliably guarantee only safety properties. For instance, although a file system interface may 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). (For a more detailed argument on this point, we refer the reader to the work of Rusbhy [14].) Third, it has been established formally in the work of Abadi and Lamport [1] that safety properties are possibly amenable to compositional reasoning, but common compositional reasoning principles such as rely-guarantee reasoning, which have been used successfully for functional verification, do not apply to liveness properties.

At a technical level, we represent security properties as formulas in a first-order temporal logic, following prior work on modeling functional correctness properties of systems.

4 Compositional Security

We present below two compositional reasoning principles that capture *relationships* among systems, adversaries and properties in our model. We explain intuitively what these principles mean and how to apply them. We emphasize that these relationships are *general*: they apply to the general class of systems, adversaries and properties described in the previous sections. They also *explain* why certain specific attacks work against specific systems and *predict* when specific systems will preserve their security properties even as adversaries come up with new attacks.

4.1 Composition Principle 1

If two system components T_1 and T_2 satisfy properties φ_1 and φ_2 respectively in isolation, does their simultaneous execution $T_1||T_2$ satisfy $\varphi_1 \wedge \varphi_2$?[2] Equivalently, assuming we have 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 is easy to see that the latter is not 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 acknowledgement and 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 where message senders can spoof their own 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 may receive an acknowledgment from T_2, without B_1 having received a payload.

Thus, not all security properties are compositional in the sense mentioned above. However, a certain useful class of 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.

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

Although local properties compose, most security properties of interest e.g., φ_{AB1} are not local. How, then, might we develop compositional proofs for security properties in general? The critical observation that allows us to proceed is that because a security property is a consequence of actions of individual threads, the proof of a security property can be factored into proofs of local properties, followed by reasoning that combines these local properties. The combination step, called *global reasoning* in the sequel, often relies on *domain-specific assumptions* about the system, i.e., assumptions that apply to all components of the system. For instance, in analysis of network protocols, the assumption that a message cannot be decrypted without proper keys is domain-specific. Such assumptions can either be axiomatic, or they can be established by an analysis of interfaces, as described later. Continuing our earlier example, suppose we make the domain-specific assumption that sender identities may not be spoofed. A proof that $\vdash T_1 : \varphi_{AB1}$ holds may be completed as follows. First, we establish that the local property φ_{B1} holds. The proof can then be completed by global reasoning: if A_1 receives a message purportedly from B_1, then because of the domain-specific assumption, the message must have been sent by B_1; then, because of φ_{B1}, we conclude that B_1 must have received a payload earlier. Interestingly, this proof remains virtually unchanged when we add the malicious thread T_2: $\vdash T_1||T_2 : \varphi_{B1}$ follows from the composition

[2]For ease of explanation, we only consider composition of two components, but the reasoning principles presented here extend easily to any number of components.

rule presented earlier and the fact $\vdash T_1 : \varphi_{B1}$ (choose $\varphi_1 = \varphi_{B1}$ and $\varphi_2 = \texttt{true}$ in the rule), whereas the global reasoning step 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.

Reasoning from Interfaces 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, any *invariant* [3] that every interface in the set preserves can be treated 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:

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

In summary, from the perspective of composition, the following style of proofs is beneficial: (1) Local reasoning: prove local properties of known threads by analysis of their programs, (2) Analysis of interfaces: prove invariants about the system by analysis of the interfaces available to threads (3) Global reasoning: combine the two by logical deduction to complete the proof. Any such proof is compositional — it is correct irrespective of what other components (possibly adversarial) exist, provided that the other components are confined to the interfaces considered in step (2).

We briefly discuss a published analysis of the widely deployed Trusted Computing technology using this method [5], and the consequent discovery of a real incompatibility between an existing standard protocol for attesting the integrity of the software stack 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 or TPM, which contains protected append-only registers to store measurements (hashes) of programs loaded into memory and a dedicated co-processor 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 integrity of the software stack on a machine to a trusted remote third-party. The protocol works by requiring each program to store, in the protected registers, the hash of any program it loads. 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 BIOS. The integrity of the software stack of a machine following this protocol can be proved to a third party by asking the co-processor to sign the contents of the protected registers with the hardware-protected key, and sending the signed hashes of loaded programs to the third party. The third-party can compare the hashes to known ones, thus validating the integrity of the software stack.

[3] Intuitively, an invariant is a property that holds throughout the execution of a system.

Note that the SRTM protocol is correct only if software that has not already been measured *cannot append* to the protected registers. Indeed, this invariant *was* true in the hardware prescribed by the initial Trusted Computing standard and, hence, this protocol was secure then. 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` may add hashes of arbitrary programs to the protected registers without actually loading them. Since the program is not measured the remote-third party obtaining the signed measurements will never detect its presence. An analysis of the protocol using the method outlined here easily discovered this incompatibility between the SRTM protocol and the `latelaunch` instruction. In the analysis, the TPM instruction-set, including `latelaunch`, were modeled as interfaces available to programs. The invariant can be established for all interfaces except `latelaunch`, thus leading to failure of a proof of correctness of SRTM with `latelaunch`, and to discovery of the actual attack.

4.2 Composition Principle 2

Although the structure of proofs presented above is very general, it does not suffice for proving inductive security properties, i.e., properties which hold at a point of time if and only if they have held at all prior points of time. Consider the following two examples:

- A file system whose access control mechanism includes a special permission `admin` which allows a user to modify permissions for other users. Suppose that, initially, only Alice and Bob have `admin` permission, and that the programs run by Alice and Bob 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 into 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 of time relies on the property having been true at all points in the past. We may prove the first example as follows: if its property does not 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 did not hold then either. A similar argument applies to the second example. Formally, these proofs proceed by *induction* over traces. The question is: can we structure these inductive proofs so that they are compositional, i.e. they are valid regardless of what other components execute simultaneously?

Fortunately, such inductive proofs can be made compositional by combining ideas from the previous section with a well-understood style of proofs called rely-guarantee reasoning [9, 11]. Suppose we wish 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 does not 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 first example above. 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 does not add the `admin` permission for anyone. Then, (1) follows for Alice and Bob's threads because they do not give the `admin` permission to anyone and for other threads because in order to change the permissions, they must have the `admin` permission i.e., φ must be previously violated, which is ruled out by the assumption in (1). (2) is the statement that if someone other than Alice or Bob has an `admin` permission, then some thread must have added that permission. This follows from an (obvious) domain-specific assumption that permissions cannot 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. (1) proves local properties, which are compositional as discussed previously. (2) is trivially compositional because all components must adhere to it. Consequently, the proof is valid irrespective of what threads execute in the system besides Alice and Bob's.

In general, any proof produced using this technique is compositional. Further, this reasoning method is compatible with reasoning about interfaces described earlier; in proving either (1) or (2) we may 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 secrecy of keys generated 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 [8]: 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 send out the AKey onto the (unprotected) network, it is 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 does not contain the unencrypted AKey relies on the fact that the incoming message does not contain the unencrypted AKey in it. The latter follows from the *inductive hypothesis* that any network adversary could not 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 does not contain the unencrypted AKey. ψ_T, for threads T of the client, KAS, and the TGS, is the property that the respective threads do not send out the AKey unencrypted. Then, the proof of property (2) is trivial, and property (1) follows from an analysis of the programs of the client, 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.

4.3 Summary

Generality Both composition principles described above are quite general. Generality of the first principle has been demonstrated by successfully proving authentication properties of network protocols [4] and integrity properties of trusted computing platforms [5]. The second principle has been applied to compositionally prove integrity properties of self-protecting data structures [8] and secrecy properties of network protocols [8, 13].

Predicting Attack Resistance Our model of interfaces can be used to predict whether or not 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 does not break the invariant will not break the security property. Dually, if even one interface does not maintain this invariant, then there is a potential attack on the system. Of course, the attack may not be real because the assumption φ_A may not be essential to the proof (there may be another proof without the assumption), but such a failure may be used as a red-flag during system design.

Explaining System Vulnerabilities A failure to complete an expected proof step may help explain why a specific system does not 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. There are many examples of such interface-level vulnerabilities. One concrete example of such a vulnerability in a security hypervisor is reported in a recent paper [7]. Another common source of attacks observed in practice arises from failure to consider certain interfaces that are available to the adversary. In this case, by omitting analysis of some interfaces, one can prove stronger invariants than ones that actually hold in the system and (incorrectly) use these invariants in proving security properties. One such example, in the context of trusted computing platforms, was mentioned earlier. Similarly, vulnerabilities have resulted from a failure to consider the direct memory access (DMA) write procedure as part of the interface available to the adversary [6].

5 Conclusion

We have presented an outline of a theory of compositional security. There are several directions for further work on this topic. 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. We expect that progress can be made on this problem by building on these prior results. Second, there is a strong need to develop and standardize domain-specific adversary models for system security. While there is existing work on such models in some domains, e.g., network protocols and trusted computing platforms, we have not yet arrived at a similar level of understanding in other important domains, such as the web platform. Finally, it is 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] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–524, 1995.

[2] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[3] Ran Canetti, Suresh Chari, Shai Halevi, Birgit Pfitzmann, Arnab Roy, Michael Steiner, and Wietse Venema. Composable security analysis of os services. Cryptology ePrint Archive, Report 2010/213, 2010. http://eprint.iacr.org/.

[4] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

[5] Anupam Datta, Jason Franklin, Deepak Garg, and Dilsun Kaynar. A logic of secure systems and its application to trusted computing. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (Oakland)*, pages 221–236, 2009.

[6] L. J. Fraim. SCOMP: A solution to the multilevel security problem. *IEEE Computer*, 16(7):26–34, 1983.

[7] Jason Franklin, Sagar Chaki, Anupam Datta, and Arvind Seshadri. Scalable parametric verification of secure systems: How to verify reference monitors without worrying about data structure size. In *Proceedings of the 31st IEEE Symposium on Security and Privacy (Oakland)*, pages 365–379, 2010.

[8] Deepak Garg, Jason Franklin, Dilsun Kaynar, and Anupam Datta. Compositional system security in the presence of interface-confined adversaries. In *Proceedings of the*

26th Conference on Mathematical Foundations of Programming Semantics (MFPS), pages 49–71, 2010.

[9] Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(4):596–619, 1983.

[10] Daryl McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.

[11] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, 1981.

[12] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 21st IEEE Symposium on Security and Privacy (Oakland)*, pages 184–200, 2001.

[13] Arnab Roy, Anupam Datta, Ante Derek, John C. Mitchell, and Seifert Jean-Pierre. Secrecy analysis in protocol composition logic. In *Formal Logical Methods for System Security and Correctness*. IOS Press, 2008.

[14] John Rushby. Kernels for safety? In T. Anderson, editor, *Safe and Secure Computing Systems*, chapter 13, pages 210–220. Blackwell Scientific Publications, 1989. (Proceedings of a Symposium held in Glasgow, October 1986).

[15] Jeannette M. Wing. A call to action: Look beyond the horizon. *IEEE Security & Privacy*, 1(6):62–67, 2003.