

Programming language methods for compositional security

Anupam Datta and
John C. Mitchell

Divide-and-conquer is an important paradigm in computer science that allows complex software systems to be built from interdependent components. However, there are widely recognized difficulties associated with developing divide-and-conquer paradigms for computer security; we do not have principles of compositional security that allow us to put secure components together to produce secure systems. The following article illustrates some of the problems and solutions we have explored in recent research on compositional security, compares them to other approaches explored in the research community, and describes important remaining challenges.

1. Introduction

Compositional security is a well-recognized scientific challenge [1]. Contemporary systems are built up from smaller components, but even if each component is secure in isolation, a system composed of secure components may not meet its security requirements—an adversary may exploit complex interactions between components to compromise security. Attacks using properties of one component to subvert another have shown up in practice in many different settings, including network protocols and infrastructure [2, 3, 4, 5, 1], web browsers and infrastructure [6, 7, 8, 9, 10], and application and systems software and hardware [11, 12, 13].

A theory of compositional security should identify *relationships* among systems, adversaries, and

properties, such that precisely defined operations over systems and adversaries preserve security properties. It should *explain* known attacks, *predict* previously unknown attacks, and *inform* design of new systems. The theory should be *general*—it should apply to a wide range of systems, adversaries, and properties. Guided by these desiderata, we initiated an investigation of compositional security in the domain of security protocols with the Protocol Composition Logic (PCL) project [14, 15, 16]. Building on these results, we then developed general secure composition principles that *transcend specific application domains* (for example, security protocols, access control systems, web



platform) in the Logic of Secure Systems (LS²) project [17]. These theories have been applied to explain known attacks, predict previously unknown attacks, and inform the design of practical protocols and software systems [12, 4, 18, 3, 19, 20, 21].

In both projects, we addressed two basic problems in compositional security: non-destructive and additive composition.

Nondestructive composition ensures that if two system components are combined, then neither degrades the security properties of the other. This is particularly complicated when system components share state.

For example, if an alternative mode of operation is added to a protocol, then some party may initiate a session in one mode and simultaneously respond to another session in another mode, using the same public key (an example of shared state) in both. Unless the modes are designed not to interfere, there may be an attack on the multimode protocol that would not arise if only one mode were possible. In a similar example, new attacks became possible when trusted computing systems were augmented with a new hardware instruction that could operate on protected registers (an example of shared state) previously accessible only through a prescribed protocol [12].

Additive composition supports a combination of system components in a way that accumulates security properties. Combining a basic key exchange protocol with an authentication mechanism to produce a protocol for authenticated key exchange

provides one example of additive composition [15]. Systematically adding cryptographic operations to basic authentication protocols to provide additional properties such as identity protection provides another example of additive composition [22].

Both additive and nondestructive compositions are important in practice. If we want a system with the positive security features of two components, A and B , we need nondestructive composition conditions to be sure that we do not lose security features we want, and we need additive composition conditions to make sure we get the advantages of A and B combined.

Before turning to a high-level presentation of technical aspects of nondestructive and additive composition in PCL and LS², we present two concrete examples that illustrate how security properties fail to be preserved under composition (that is, both examples are about the failure of nondestructive composition). We also compare our composition methods to three related approaches—compositional reasoning for correctness properties of systems [23, 24], the universal composability framework [25, 26], and a refinement type system for compositional type-checking of security protocols [27]. Finally, we describe directions for future work.

2. Two examples

While these protocol examples are contrived, the phenomena they illustrate are not: It is possible for one component of a system to expose an interface to the adversary that does not affect its own security but compromises the security of other components. Later, we will describe two general principles of compositional security that could be used to design security protocols and other kinds of secure software systems while avoiding the kind of insecure interaction illustrated by these examples.

Example 1: Authentication failure. The following two protocols use digital signatures. The first protocol provides one-way authentication when used in isolation; however, this property is not preserved when the second protocol is run concurrently.

- ▶ *Protocol 1.1.* Alice generates a fresh random number r and sends it to Bob. Upon receiving such a message, Bob replies to the sender of the message (as recorded in the message) with his signature over the fresh random number and

the sender’s name—that is, if Bob receives the message with the random number r from sender A , then Bob replies with his signature over r and A . This protocol guarantees a form of one-way authentication: After sending the first message to Bob and then receiving Bob’s second message, Alice is guaranteed that Bob received the first message that she sent to him and then sent the second message and intended it for her.

- ▶ *Protocol 1.2.* Upon receiving any message m , Bob signs it with his private signing key and sends it out on the network.

When the two protocols are run concurrently, protocol 1.1 no longer provides one-way authentication: Alice cannot be certain that Bob received her first message and intended the signed message for her as part of the execution of this protocol; it could very well be that Bob produced the signature as part of protocol 1.2 in response to an adversary M who intercepted Alice’s message and used it to start a session of protocol 1.2 with Bob.

Example 2: Secrecy failure. Using network protocols as an illustration, here are two secure, unidirectional protocols for communication between Alice and Bob. Both involve public key cryptography, in which two different keys are used for encryption and decryption, and the encryption key may be distributed publicly.

- ▶ *Protocol 2.1.* In this protocol, for communication from Alice to Bob, Alice sends a message to Bob by encrypting it with Bob’s public encryption key. As part of each message, in order to make our example illustrate the general point, Alice also reveals her secret decryption key, making public-key encryption to Alice insecure.
- ▶ *Protocol 2.2.* This protocol is the same as the previous one (that is, protocol 2.1), but in reverse: Bob communicates to Alice by encrypting messages using Alice’s public key and revealing his own private decryption key.

Both protocol 2.1 and 2.2 are secure when used by themselves: If Bob sends Alice a message encrypted with Alice’s public key, then only Alice can decrypt and read the message. However, it should be clear that composing these two protocols to communicate between Alice and Bob in both directions is completely insecure because when Alice sends Bob a message,

she leaks her private key, and when Bob communicates to Alice, he leaks his private key. After at least one message in each direction, both public keys have been leaked and any eavesdropper on the network can decrypt and read all the messages.

3. Two principles of secure composition

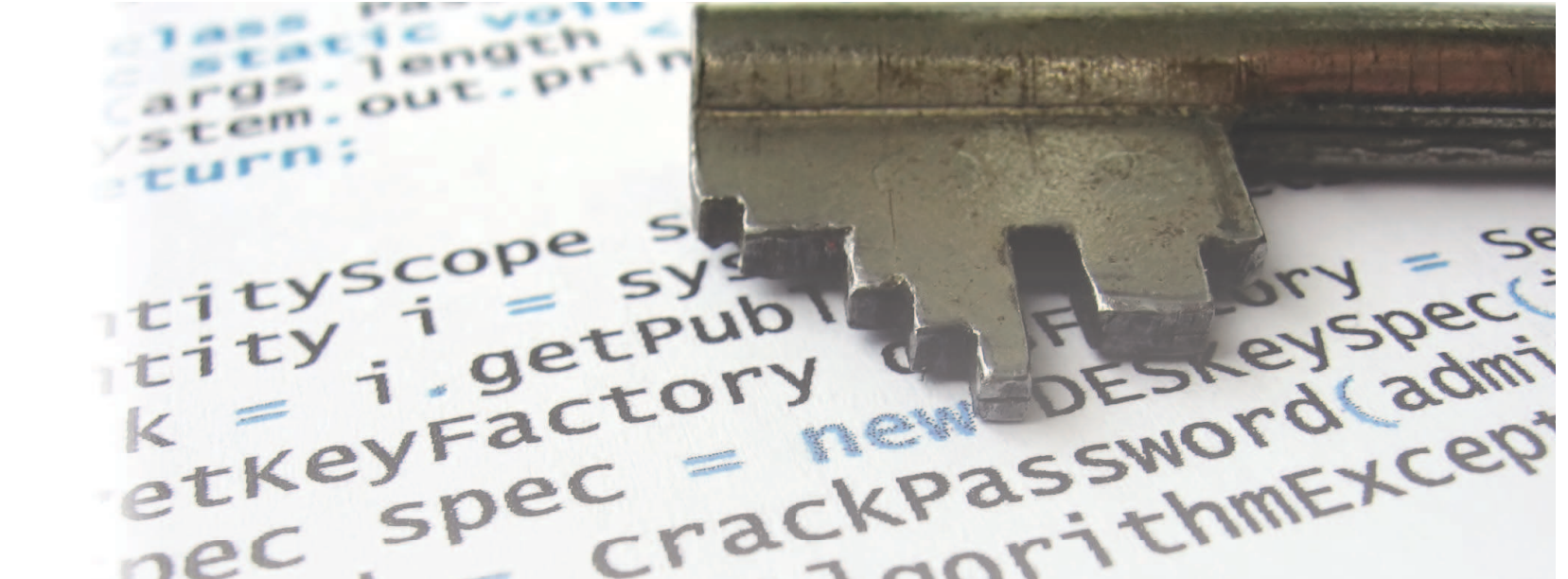
In the following, we describe two principles of secure composition, and we use these principles to explain the examples of insecure composition in the previous section.

3.1. Principle 1: Preserving invariants of system components

The central idea behind this principle is that the security property of a system component is preserved under composition if every other component respects invariants used in the proof of security of the component in the face of attack. In example 1, the only relevant invariant for the authentication property of protocol 1.1 is of the following form: “If an honest^a principal signs a message of the form $\langle r, A \rangle$, then he must have previously received r in a message with A as the identifier for the sender.” This invariant is not preserved by protocol 1.2, as demonstrated by the attack described in the previous section, leading to a failure of nondestructive composition.

To illustrate the generality of this principle, we briefly discuss a published analysis of the widely deployed Trusted Computing Group (TCG) technology using this principle [12], and we discuss 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 (that is, 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 integrity of the software stack on a machine to a trusted remote third

a. A principal is honest if he does not deviate from the steps of the protocol.



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 basic input/output system (BIOS). The integrity of the software stack of a machine 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 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.

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 TCG 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 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 leading to discovery of the actual attack.

This composition principle is related to the form of assume-guarantee reasoning initially proposed by Jones for reasoning about correctness properties of concurrent programs [23]. However, one difference is that, in contrast to Jones' work, we consider preservation of properties of system components under composition in the presence of an active adversary whose exact program (or invariants) is not known. After sketching the technical approach in the next sections, we will explain how we address this additional complexity.

3.2. Principle 2: Secure rely-guarantee reasoning

Inductive security properties (that is, properties which hold at a point of time if and only if they have held at all prior points of time) require a different form of compositional reasoning that builds on prior work on rely-guarantee reasoning for correctness properties of programs [23, 24].

Suppose we wish to prove that property φ holds at all times. First, we identify a set $S = \{T_1, \dots, T_n\}$ of trusted components relevant to the property and local properties $\Psi_{T_1}, \dots, \Psi_{T_n}$ of these components, satisfying the following conditions:

- (1) If φ holds at all time points strictly before any given time point, then each of $\Psi_{T_1}, \dots, \Psi_{T_n}$ holds at the given time point.
- (2) If φ does not hold at any time, then at least one of $\Psi_{T_1}, \dots, \Psi_{T_n}$ 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 return to example 2 to illustrate the application of this principle. In order to prove the secrecy of the encrypted message, it is necessary to prove that the private decryption key is known only to the associated party. If protocol 2.1 (or protocol 2.2) were to run in isolation, the relevant decryption key would indeed be known only to the associated party (Alice or Bob). This can be proved using the rely-guarantee reasoning technique described above and noting that the recipient of the encrypted message never sends out his or her private decryption key and that the other party cannot send it out (assuming that it has not already been sent out). However, when the two protocols are composed in parallel, the proof no longer works because the sender in one protocol is the recipient in the other; thus, we can no longer prove that the recipient's private decryption key is not sent out on the network. Indeed, the composition attack arises precisely because the recipient's private decryption key is sent out on the network.

Another application of the rely-guarantee technique is in proofs of secrecy of symmetric keys generated in network protocols. We explain one instance here—proving that the so called authentication key (AKey) generated during the Kerberos V protocol (a widely used industry standard) becomes known only to three protocol participants [17, 18]: 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 client's outgoing message does not contain the unencrypted AKey because 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 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. The properties Ψ_r , for components

T of the client, KAS, and the TGS model the requirement that the respective components do not send out the AKey unencrypted. Then, the proof of condition (2) of the rely-guarantee framework is trivial, and condition (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, that is, the proof is compositional.

4. Protocol Composition Logic

Protocol Composition Logic (PCL) [14, 15, 16] is a formal logic for proving security properties of network protocols that use public and symmetric key cryptography. The system has several parts:

- ▶ **A simple programming language for defining protocols by writing programs for each role of the protocol.** For example, the secure sockets layer (SSL) protocol can be modeled in this language by writing two programs—one for the client role and one for the server role of SSL. Each program is a sequence of actions, such as sending and receiving messages, decryption, and digital signature verification. The operational semantics of the programming language define how protocols execute concurrently with a symbolic adversary (sometimes referred to as the Dolev-Yao adversary) that controls the network but cannot break the cryptographic primitives.
- ▶ **A pre/postcondition logic for describing the starting and ending security conditions for protocol.** For example, a precondition might state that a symmetric key is shared by two agents, and a postcondition might state that a new key exchanged using the symmetric key for encryption is only known to the same two agents.
- ▶ **Modal formulas, denoted $\theta[P]_x \phi$, for stating that if a precondition θ holds initially, and a protocol thread X completes the steps P , then the postcondition ϕ will be true afterwards irrespective of concurrent actions by other agents and the adversary.** Typically, security properties of protocols are specified in PCL using such modal formulas.

- ▶ **A formal proof system for deriving true modal formulas about protocols.** The proof system consists of axioms about individual protocol actions and inference rules that yield assertions about protocols composed of multiple steps.

One of the important ideas in PCL is that although assertions are written only using the steps of the protocol, the logic is sound in a strong sense: Each provable assertion involving a sequence of actions holds in any protocol run containing the given actions and arbitrary additional actions by a malicious adversary. This approach lets us prove security properties of protocols under attack while reasoning only about the actions of honest parties in the protocol, thus significantly reducing the size of protocol proofs in comparison to other proof methods, such as Paulson's Inductive Method [28].

Intuitively, additive combination is achieved using modal formulas of the form $\theta[P]_A \phi$. For example, the precondition θ might assert that A knows B 's public key, the actions P allow A to receive a signed message and verify B 's signature, and the postcondition ϕ may say that B sent the signed message that A received. The importance of modal formulas with before-after assertions is that we can combine assertions about individual protocol steps to derive properties of a sequence of steps: If $\phi[P]_A \psi$ and $\psi[P']_A \theta$, then $\phi[PP']_A \theta$. For example, an assertion assuming that keys have been successfully distributed can be combined with steps that do key distribution to prove properties of a protocol that distributes keys and uses them.

We ensure one form of nondestructive combination using invariance assertions, capturing the first composition principle described in Section 3. The central assertion in our reasoning system, $\Gamma \vdash \phi[P]_A \psi$, says that in any protocol satisfying the invariant Γ , the before-after assertion $\phi[P]_A \psi$ holds in any run (regardless of any actions by any dishonest attacker). Typically, our invariants are statements about principals that follow the rules of a protocol, as are the final conclusions. For example, an invariant may state that every honest principal maintains secrecy of its keys, where honest means simply that the principal only performs actions that are given by the protocol. A conclusion in such a protocol may be that if Bob is *honest* (so no one else knows his key), then after Alice sends and receives certain messages, Alice knows that she has communicated with Bob. Nondestructive combination occurs

when two protocols are combined and neither violates the invariants of the other.

PCL also supports a specialized form of secure rely-guarantee reasoning about secrecy properties, capturing the second composition principle in Section 3. In order to prove that the network is safe (that is, all occurrences of the secret on the network appear under encryption with a set of keys \mathbf{K} not known to the adversary), the proof system requires us to prove that assuming that the network is safe, all honest agents only send out “safe” messages, that is, messages from which the secret cannot be extracted without knowing the keys in the set \mathbf{K} [18].

These composition principles have been applied to prove properties of a number of industry standards including SSL/TLS, IEEE 802.11i, and Kerberos V5.

5. Logic of Secure Systems

The Logic of Secure Systems (LS²) (initially presented in [12]) builds on PCL to develop related composition principles for secure systems that perform network communication and operations on local shared memory as well as on associated adversary models. These principles have been applied to study industrial trusted computing system designs. The study uncovered an attack that arises from insecure composition between two remote attestation protocols (see [12] for details). A natural scientific question to ask is whether one could build on these results to develop general secure composition principles that transcend specific application domains, such as network protocols and trusted computing systems. Subsequent work on LS² [17], which we turn to next, answers exactly this question.

Two goals drove the development of LS². First, we posit that a general theory of secure composition must enable one to flexibly model and parametrically reason about different classes of adversaries. To develop such a theory, we view a trusted system in terms of the interfaces its various components expose: Larger trusted components are built by connecting interfaces in the usual ways (client-server, call-return, message-passing, etc.). The adversary is confined to some subset of the interfaces, but its program is unspecified and can call those interfaces in ways that are not known a priori. Our focus on interface-confined adversaries thus provides a generic way to model different classes of

adversaries in a compositional setting. For example, in virtual machine monitor-based secure systems, we model an adversarial guest operating system by confining it to the interface exposed by the virtual machine monitor. Similarly, adversary models for web browsers, such as the gadget adversary (an attractive vector for malware today that leverages properties of Web 2.0 sites), can be modeled by confining the adversary to the read and write interfaces for frames guarded by the same-origin policy as well as by frame navigation policies [7]. The network adversary model considered in prior work on PCL and the adversary against trusted computing systems considered in the initial development of LS² are also special cases of this interface-confined adversary model. 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.

Our second goal was to develop compositional reasoning principles for a wide range of classes of interconnected systems and associated interface-confined adversaries that are described using a rich logic. The approach taken by LS² uses a logic of program specifications, employing temporal operators to express not only the states and actions at the beginning and end of a program, but also at points in between. This expressiveness is crucial because many security properties of interest, such as integrity properties, are safety properties [29]. LS² supports the two principles of secure composition discussed in the previous section in the presence of such interface-confined adversaries. The first principle follows from a proof rule in the logic, and the second principle follows from first-order reasoning in the logic. We refer the interested reader to our technical paper for details [17].

6. Related work

We compare our approach to three related approaches—compositional reasoning for correctness properties of systems [23, 24], the Universal Composability

(UC) framework [25, 26], and a refinement type system for compositional type-checking of security protocols [27].

The secure composition principles we developed are related to prior work on rely-guarantee reasoning for correctness properties of programs [23, 24]. However, the prior work was developed for a setting in which all programs are known. In computer security, however, it is unreasonable to assume that the adversary’s program is known a priori; rather, we model adversaries as arbitrary programs that are confined to certain system interfaces as explained earlier. We prove invariants about trusted programs and system interfaces that hold irrespective of concurrent actions by other trusted programs and the adversary. This additional generality, which is crucial for the secure composition principles, is achieved at a technical level using novel invariant rules. These rules allow us to conclude that such invariants hold by proving assertions of the form $\theta[P]_x \phi$ over trusted programs or system interfaces; note that because of the way the semantics of the modal formula is defined, the invariants hold irrespective of concurrent actions by other trusted programs and the adversary, although the assertion only refers to actions of one thread X .

Recently, Bhargavan et al. developed a type system to modularly check interfaces of security protocols, implemented the system, and applied it to analysis of secrecy properties of cryptographic protocols [27]. Their approach is based on refinement types (that is, ordinary types qualified with logical assertions), which can be used to specify program invariants and pre- and postconditions. Programmers annotate various points in the model with assumed and asserted facts. The main safety theorem states that all programmer defined assertions are implied by programmer assumed facts in a well-typed program.

However, a semantic connection between the program state and the logical formulas representing assumed and asserted facts is missing. In contrast, we prove that the inference systems of our logics of programs (PCL and LS²) are sound with respect to trace semantics of the programming language. Our logic of programs may provide a semantic foundation for the work of Bhargavan et al. and, dually, the implementation in that work may provide a basis for


mechanizing the formal system in our logics of programs. Bhargavan et al.'s programming model is more expressive than ours because it allows higher-order functions. We intend to add higher-order functions to our framework in the near future.

While all the approaches previously discussed involve proving safety properties of protocols and systems modeled as programs, an alternative approach to secure composition involves comparing the real protocol (or system) whose security we are trying to evaluate to an ideal functionality that is secure by construction and prove that the two are equivalent in a precise sense. Once the equivalence between the real protocol and the ideal functionality is established, the composition theorem guarantees that any larger system that uses the real protocol is equivalent to the system where the real protocol is replaced by the ideal functionality.

This approach has been taken in the UC framework for cryptographic protocols [25, 26] and is also related to the notion of observational equivalence and simulation relations studied in the programming languages and verification literature [30, 31]. When possible, this form of composition result is indeed very strong: Composition is guaranteed under no assumptions about the environment in which a component is used. However, components that share state and rely on one another to satisfy certain assumptions about how that state is manipulated cannot be compositionally analyzed using this approach; the secure rely-guarantee principle we develop is better suited for such analyses. One example is the compositional security analysis of the Kerberos protocol that proceeds from proofs of its constituent programs [18].

7. Future work

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, while sequential composition of secure systems is

an important step forward, a general treatment of additive composition that considers other forms of composition is still missing. Third, 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. Finally, a quantitative theory of compositional security that supports analysis of systems built from components that are not perfectly secure would be a significant result. 


About the authors

Anupam Datta is an assistant research professor at Carnegie Mellon University. Dr. Datta's research focuses on foundations of security and privacy. He has made contributions toward advancing the scientific understanding of security protocols, privacy in organizational processes, and trustworthy software systems. Dr. Datta has coauthored a book and over 30 publications in conferences and journals on these topics. He serves on the Steering Committee of the IEEE Computer Security Foundations Symposium (CSF), and has served as general chair of CSF 2008 and as program chair of the 2008 Formal and Computational Cryptography Workshop and the 2009 Asian Computing Science Conference. Dr. Datta obtained MS and PhD degrees from Stanford University and a BTech from the Indian Institute of Technology, Kharagpur, all in computer science.

John C. Mitchell is the Mary and Gordon Crary Family Professor in the Stanford Computer Science Department. His research in computer security focuses on trust management, privacy, security analysis of network protocols, and web security. He has also worked on programming language analysis and design, formal methods, and other applications of mathematical logic to computer science. Professor Mitchell is currently involved in the multiuniversity Privacy, Obligations, and Rights in Technology of Information Assessment (PORTIA) research project to study privacy concerns in databases and information processing systems, and the National Science Foundation Team for Research in Ubiquitous Secure Technology (TRUST) Center.

References

- [1] Wing JM. A call to action: Look beyond the horizon. *IEEE Security & Privacy*. 2003;1(6):62–67. DOI: 10.1109/MSECP.2003.1253571
- [2] Asokan N, Niemi V, Nyberg K. Man-in-the-middle in tunnelled authentication protocols. In: Christianson B, Crispo B, Malcolm JA, Roe M, editors. *Security Protocols 11th International Workshop, Cambridge, UK, April 2-4, 2003, Revised Selected Papers*. Berlin (Germany): Springer-Verlag; 2005. p. 28–41. ISBN 13: 978-354-0-28389-8
- [3] Kuhlman D, Moriarty R, Braskich T, Emeott S, Tripunitara M. A correctness proof of a mesh security architecture. In: *Proceedings of the 21st IEEE Computer Security Foundations Symposium*; Jun 2008; Pittsburgh, MA. p. 315–330. DOI: 10.1109/CSF.2008.23
- [4] Meadows C, Pavlovic D. Deriving, attacking and defending the GDOI protocol. In: *Proceedings of the Ninth European Symposium on Research in Computer Security*; Sep 2004; Sophia Antipolis, France. p. 53–72. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.3254&rep=rep1&type=pdf>
- [5] Mitchell JC, Shmatikov V, Stern U. Finite-state analysis of SSL 3.0. In: *Proceedings of the Seventh Conference on USENIX Security Symposium*; Jan 1998; San Antonio, TX. p. 16. Available at: <http://www.usenix.org/publications/library/proceedings/sec98/mitchell.html>
- [6] Barth A, Jackson C, Mitchell JC. Robust defenses for cross-site request forgery. In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*; Oct 2008; Alexandria, VA. p. 75–88. DOI: 10.1145/1455770.1455782
- [7] Barth A, Jackson C, Mitchell JC. Securing frame communication in browsers. In: *Proceedings of the 17th USENIX Security Symposium*; Jul 2008; San Jose, CA. p. 17–30. Available at: http://www.usenix.org/events/sec08/tech/full_papers/barth/barth.pdf
- [8] Chen S, Mao Z, Wang YM, Zhang M. Pretty-bad-proxy: An overlooked adversary in browsers' HTTPS deployments. In: *Proceedings of the 30th IEEE Symposium on Security and Privacy*; May 2009; Oakland, CA. p. 347–359. DOI: 10.1109/SP.2009.12
- [9] Jackson C, Barth A. ForceHTTPS: Protecting high-security web sites from network attacks. In: *Proceedings of the 17th International Conference on World Wide Web*; Apr 2008; Beijing, China. p. 525–534. Available at: <http://www2008.org/papers/pdf/p525-jacksonA.pdf>
- [10] Jackson C, Barth A, Bortz A, Shao W, Boneh D. Protecting browsers from DNS rebinding attacks. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*; Oct 2007; Alexandria, VA. p. 421–431. DOI: 10.1145/1315245.1315298
- [11] Cai X, Gui Y, Johnson R. Exploiting Unix file-system races via algorithmic complexity attacks. In: *Proceedings of the 30th IEEE Symposium on Security and Privacy*; May 2009; Oakland, CA; p. 27–41. DOI: 10.1109/SP.2009.10
- [12] Datta A, Franklin J, Garg D, Kaynar D. A logic of secure systems and its application to trusted computing. In: *Proceedings of the 30th IEEE Symposium on Security and Privacy*; May 2009; Oakland, CA. p. 221–236. DOI: 10.1109/SP.2009.16
- [13] Tsafirir D, Hertz T, Wagner D, Da Silva D. Portably solving file TOCTTOU races with hardness amplification. In: *Proceedings of the Sixth USENIX Conference on File and Storage Technologies*; Feb 2008; San Jose, CA. p. 1–18. Available at: <http://www.usenix.org/events/fast08/tech/tsafirir.html>
- [14] Datta A, Derek A, Mitchell JC, Pavlovic D. A derivation system and compositional logic for security protocols. *Journal of Computer Security*. 2005;13(3):423–482. Available at: <http://seclab.stanford.edu/pcl/papers/ddmp-jcs05.pdf>
- [15] Datta A, Derek A, Mitchell JC, Roy A. Protocol composition logic (PCL). *Electronic Notes in Theoretical Computer Science*. 2007;172:311–358. DOI: 10.1016/j.entcs.2007.02.012
- [16] Durgin N, Mitchell JC, Pavlovic D. A compositional logic for proving security properties of protocols. *Journal of Computer Security*. 2003;11(4):677–721. Available at: <http://www-cs-students.stanford.edu/~nad/papers/comp-jcs205.pdf>
- [17] Garg D, Franklin J, Kaynar DK, Datta A. Compositional system security with interface-confined adversaries. *Electronic Notes in Theoretical Computer Science*. 2010;265:49–71. DOI: 10.1016/j.entcs.2010.08.005
- [18] Roy A, Datta A, Derek A, Mitchell JC, Seifert JP. Secrecy analysis in protocol composition logic. In: Okada M, Satoh I, editors. *Advances in Computer Science – ASIAN 2006: Secure Software and Related Issues, 11th Asian Computing Science Conference, Tokyo, Japan, December 6-8, 2006*. Berlin (Germany): Springer-Verlag; 2007. p. 197–213.
- [19] Butler KRB, McLaughlin SE, McDaniel PD. Kells: A protection framework for portable data. In: *Proceedings of the 26th Annual Computer Security Applications Conference*; Dec 2010; Austin, TX. p. 231–240. DOI: 10.1145/1920261.1920296
- [20] Kannan J, Maniatis P, Chun B. Secure data preservers for web services. In: *Proceedings of the Second USENIX Conference on Web Application Development*; Jun 2011; Portland, OR. p. 25–36. Available at: http://www.usenix.org/events/webapps11/tech/final_files/Kannan.pdf

- 
- [21] He C, Sundararajan M, Datta A, Derek A, Mitchell JC. A modular correctness proof of IEEE 802.11i and TLS. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security*; Nov 2005; Alexandria, VA. p. 2–15. DOI: 10.1145/1102120.1102124
- [22] Datta A, Derek A, Mitchell JC, Pavlovic D. Abstraction and refinement in protocol derivation. In: *Proceedings of 17th IEEE Computer Security Foundations Workshop*; Jun 2004; Pacific Grove, CA. p. 30–45. DOI: 10.1109/CSFW.2004.1310730
- [23] Jones CB. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*. 1983;5(4):596–619. DOI: 10.1145/69575.69577
- [24] Misra J, Chandy KM. Proofs of networks of processes. *IEEE Transactions on Software Engineering*. 1981;7(4):417–426. DOI: 10.1109/TSE.1981.230844
- [25] Canetti R. Universally composable security: A new paradigm for cryptographic protocols. In: *Proceedings of the 42nd IEEE Symposium on the Foundations of Computer Science*; Oct 2001; Las Vegas, NV. p. 136–145. DOI: 10.1109/SFCS.2001.959888
- [26] Pfitzmann B, Waidner M. A model for asynchronous reactive systems and its application to secure message transmission. In: *IEEE Symposium on Security and Privacy*; May 2001; Oakland, CA. p. 184–200. DOI: 10.1109/SECPRI.2001.924298
- [27] Bhargavan K, Fournet C, Gordon AD. Modular verification of security protocol code by typing. In: *Proceedings of the 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*; Jan 2010; Madrid, Spain. p. 445–456. DOI: 10.1145/1706299.1706350
- [28] Paulson L. Proving properties of security protocols by induction. In: *Proceedings of 10th IEEE Computer Security Foundations Workshop*; Jun 1997; Rockport, MA. p. 70–83. DOI: 10.1109/CSFW.1997.596788
- [29] Alpern B, Schneider FB. Recognizing safety and liveness. *Distributed Computing*. 1987;2(3):117–126. DOI: 10.1007/BF01782772
- [30] Canetti R, Cheung L, Kaynar DK, Liskov M, Lynch NA, Pereira O, Segala R. Time-bounded task-PIOAs: A framework for analyzing security protocols. In: *Proceedings of the 20th International Symposium on Distributed Computing*; Sep 2006; Stockholm, Sweden. p. 238–253. DOI: 10.1007/11864219_17
- [31] Küsters R, Datta A, Mitchell JC, Ramanathan A. On the relationships between notions of simulation-based security. *Journal of Cryptology*. 2008;21(4):492–546. DOI: 10.1007/s00145-008-9019-9