Secure Embedded Systems

Michael Vai, David J. Whelihan, Benjamin R. Nahill, Daniil M. Utin, Sean R. O'Melia, and Roger I. Khazan

Developers seek to seamlessly integrate cyber security within U.S. military system software. However, added security components can impede a system's functionality. System developers need a well-defined approach for simultaneously designing functionality and cyber security. Lincoln Laboratory's secure embedded system co-design methodology uses a security coprocessor to cryptographically ensure system confidentiality and integrity while maintaining functionality.

Department of Defense (DoD) systems, e.g., computer networks, are increasingly the targets of deliberate, sophisticated cyber attacks. To assure successful missions, military systems must be secured to perform their intended functions, prevent attacks, and operate while under attack. The DoD has further directed that cyber security technology must be integrated into systems because it is too expensive and impractical to secure a system after it has been designed [1]. To address this directive, Lincoln Laboratory is using a co-design approach to systems that meet both security and functionality requirements. The Laboratory is at the research and development forefront of system solutions for challenging critical missions, such as those to collect, process, and exchange sensitive information. Many of Lincoln Laboratory's prototype systems must be designed with security in mind so that they can be quickly brought into compliance with the DoD's cyber security requirements and support field tests and technology transfer.

Many DoD systems require the use of embedded computing. An embedded computer system is designed for a dedicated function, in contrast to a general-purpose computer system, e.g., a desktop computer, which is designed for multiple functions [2]. An ideal design for an embedded system optimizes performance, e.g., small form factor, low power consumption, and high throughput, while providing the specific functionality demanded by the system's purpose, i.e., its mission. Developers must also determine the embedded system's security requirements according to mission objectives and a concept of operations (CONOPS). In general, security should be robust

enough to prevent attacks, ensuring that a system can successfully support a mission. Developers may need to enable a system to continue functioning, albeit with possibly degraded capabilities, when security fails. The design of security for an embedded system is challenging because security requirements are rarely accurately identified at the start of the design process. As a result, embedded systems' engineers tend to focus on well-understood functional capabilities rather than on stringent security requirements. In addition, engineers must provide security that causes minimal impacts on a system's size, weight, and power (SWaP), usability, cost, and development schedule.

To meet these challenges, we established a secure embedded system development methodology. When securing a system, we strive to achieve three goals: confidentiality, integrity, and availability, which are often referred to as the CIA triad for information security. The CIA triad is defined for embedded systems as follows:

- Confidentiality ensures that an embedded system's critical information, such as application code and surveillance data, cannot be disclosed to unauthorized entities.
- Integrity ensures that adversaries cannot alter system operation.
- Availability assures that mission objectives cannot be disrupted.

In this article, we use the example of a hypothetical secure unmanned aircraft system (UAS) to illustrate how we use cryptography to ensure confidentiality and integrity. Using this example, we demonstrate the identification of potential attack targets by considering the CONOPS, the development of countermeasures to these attacks, and the design and implementation of a cryptography-based security architecture. Because cryptography does not directly enable availability, we also provide insight into ongoing research that extends our methodology to achieve the resilience required to improve the availability of embedded systems.

Challenges in Securing Embedded Systems

An embedded system will provide very little, if any, SWaP allowance for security; thus, security must not impose excessive overheads on the protected system. While the DoD has some of the most demanding applications in terms of throughput and SWaP, it no longer drives the development of processor technology. Therefore, security technologies must be compatible with embedded systems that use commercial off-the-shelf (COTS) processor hardware platforms that the DoD can easily adopt.

As military electronic systems continue to increase in sophistication and capability, their cost and development time also grow. Each year, the DoD acquires and operates numerous embedded systems, ranging from intelligence, surveillance, and reconnaissance sensors to electronic warfare and electronic signals intelligence systems. Depending on their CONOPS, embedded systems have different security requirements. Methodologies for securing embedded systems must be customizable to meet CONOPS needs.

To meet application-specific requirements while also reducing technology costs and development time, developers have started to use open-systems architectures (OSA). Because OSAs use nonproprietary system architectural standards in which various payloads can be shared among various platforms, technology upgrades are easy to access and implement. The DoD has thus directed all DoD agencies to adopt OSA in electronic systems [3]. However, adding security to OSA could interfere with its openness. As most current security approaches are ad hoc, proprietary, and expensive, they are incompatible with OSA principles, especially when each payload developer individually implements and manages the payload security. Therefore, developing a system-level secure embedded system architecture that will seamlessly work with various OSA components is a challenge.

Design Process

Embedded system CONOPS are developed from mission objectives and are used to derive both functional and security requirements. Researchers create, evaluate, and implement an initial system design, codeveloping functionality and security while minimizing security interference during functionality testing by decoupling security and functionality requirements. Several design iterations may be required before the mission objectives are met. Figure 1 captures the ideal process of designing a secure embedded system; the steps dedicated to security are highlighted in green.

To illustrate the secure embedded system design process, we use the design of a hypothetical UAS for a video surveillance application. The CONOPS of this example UAS application is as follows: At startup, the

UAS loads its long-term credentials for identification and authentication purposes. Mission-specific information—e.g., software, firmware, and data—is loaded into the respective memories. The system is then booted up and prepared for mission execution.

Figure 2 illustrates the UAS embedded system in its execution phase. Under the command of a ground control station, the UAS takes off, flies to its destination, and then collects video data. Video data containing target information are encrypted and broadcast to authorized ground stations (GT1 and GT2) via a radio. Raw video data are also saved for further processing after the UAS lands. When the UAS is shut down, both raw and processed video data are considered sensitive and must be saved securely. Any persistent-state data, such as long-term credentials, must also be protected.

Figure 3 shows a high-level functional architecture initially designed for the example UAS embedded system. The architecture consists of a central processing unit (CPU) and a field-programmable gate array (FPGA) interconnected with a backplane network. The FPGA typically performs advanced video signal processing (e.g., for target detection and identification). The CPU handles command-and-control communications received from the ground control station and manages information (e.g., for target tracking).

Processing elements, such as the CPU and FPGA, must be chosen to securely deliver the UAS functionality requirements. This UAS application involves sophisticated signal processing and requires high throughput (measured by the number of floating-point operations per second) with a stringent SWaP allowance.

To support a complicated signal processing algorithm, the CPU needs a large memory and storage capacity. A popular mainstream processor likely has a variety of COTS software libraries that can be used in application development, but it may not have the security features desired for the CONOPS. On the other hand, a secure processor with built-in security features may simplify system development but may not possess the appropriate processing power or support the large memory space required for the application. We must consider system openness and upgradability before choosing a secure processor over a mainstream CPU.

Many popular FPGAs are built with embedded security features [4]. Developers should select these devices on the basis of their ability to encrypt and authenticate configuration bitstreams, incorporate security monitors to detect attacks, and erase decryption keys (a process known as zeroization) to protect critical information when attacks are detected.

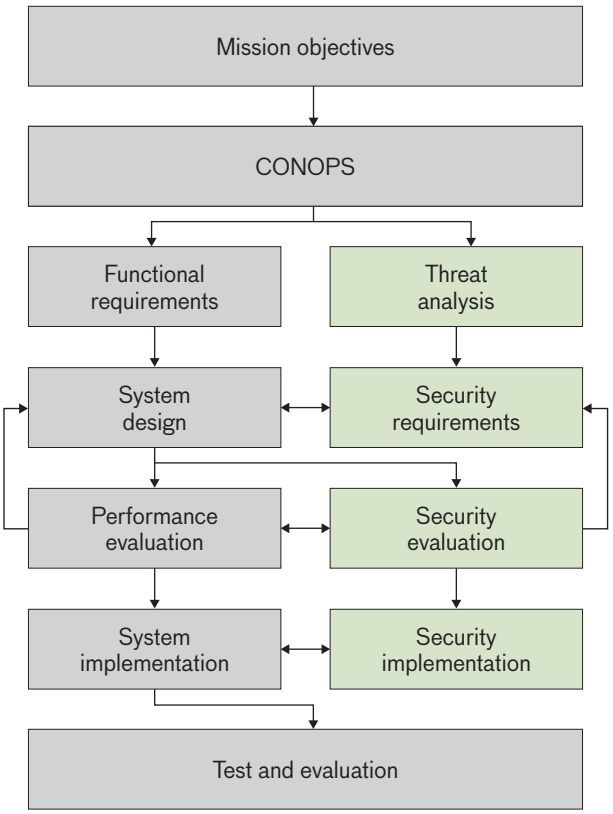

FIGURE 1. In an ideal secure embedded system design process, functionality (gray) and security (green) are co-designed, yet they are appropriately decoupled during testing so that security does not interfere with functionality. This co-design is often difficult to achieve because functionality and security are two very different disciplines.

Threat Analysis

The first step in designing a secure system is to analyze the potential attacks that the system may be subjected to when deployed. Adversaries seek to sabotage and develop countermeasures against U.S. missions, so the CONOPS determines not only functional requirements but also potential adversary attacks. The attacks depend on the adversary's capability (e.g., a nation state's sophisticated knowledge) and objectives (e.g., to exfiltrate information).

In the UAS example, we assume that there is a high probability of equipment loss resulting from the small size of the UAS and its operation in hostile areas. The examples of UAS attack targets in Figure 4 portray three logical

attack surfaces—boot process, system data, and software—and one physical attack surface, its physical system, that adversaries may attack to exfiltrate information.

During the CPU boot process, a secure system must establish a root of trust, which consists of hardware and software components that are inherently trusted, to protect and authenticate software components. Current practice uses the trusted platform module (TPM), an international standard secure processor that facilitates secure cryptographic key generation, remote attestation, encryption, decryption, and sealed storage [5]. Each TPM chip includes a unique secret key, allowing the chip to perform platform and hardware device authentication.

When creating the TPM, developers make a number of compromises that address cost and privacy concerns to ensure commercial adoptability of the module by vendors. The TPM must be inexpensive and cause as little disruption to the processing architecture as possible. Consumer privacy concerns dealing with user identification force module usage to be an optional and passive part of a processing system's operations. These compromises lead to a low-performance module that lacks adequate physical protection. In the "Architecture and Enabling Technologies" section, we will explain Lincoln Laboratory's security coprocessor that is equipped with a physical unclonable function, which was developed to address the TPM security inadequacy in tactical operations.

Despite a system's incorporation of an effective TPM, adversaries may exploit latent vulnerabilities within an authorized software component to access critical data or gain control of the platform itself. Even authorized users could deliberately or negligently introduce threats onto a system via untrusted software (e.g., malware) or unwanted functionality via third-party intellectual property.

A secure system must be designed to prevent compromised software from giving an attacker unrestricted system access. Some developers are starting to address access issues on commercial systems. For example, software developers use separation kernels to establish and isolate individual computing processes, control information flow between the processes, and prevent unauthorized information access. On the hardware side, researchers are developing architectures that enforce isolations between processing threads executing on the same processor [6].

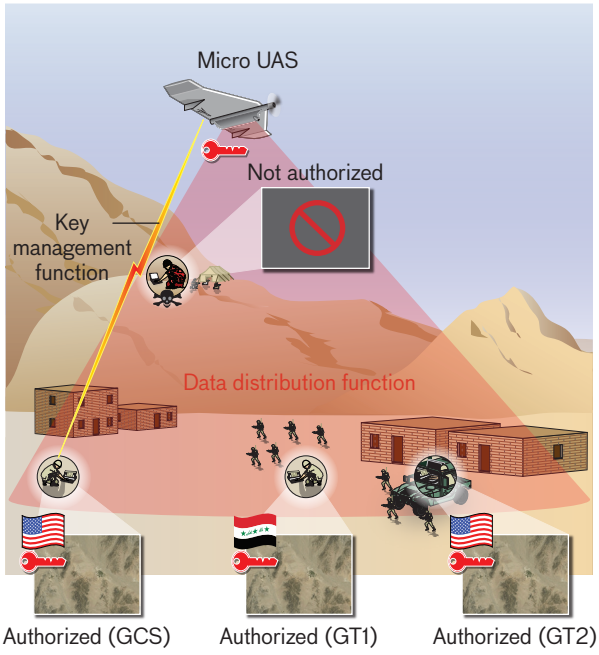

FIGURE 2. In this example of an unmanned aircraft system (UAS) application in its execution phase, the intelligence collected by the UAS needs to be shared by coalition partners yet protected from adversaries. Cryptography is the key technology enabling this operation.

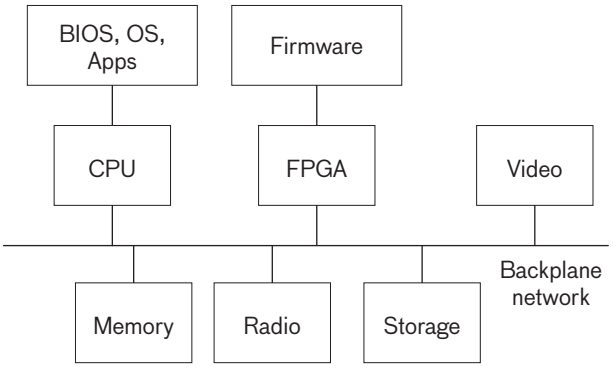

FIGURE 3. This example of an unmanned aircraft system's embedded system functional architecture includes the central processing unit (CPU) that is supplied with a basic input/output system (BIOS), operating system (OS), and mission-specific application code (Apps). The field-programmable gate array (FPGA) has its configuration stored in a firmware memory. In addition to a video camera payload, the system has a random-access memory, a hard drive for storage, and a radio, all of which are accessible by the CPU and/or FPGA through a backplane network.

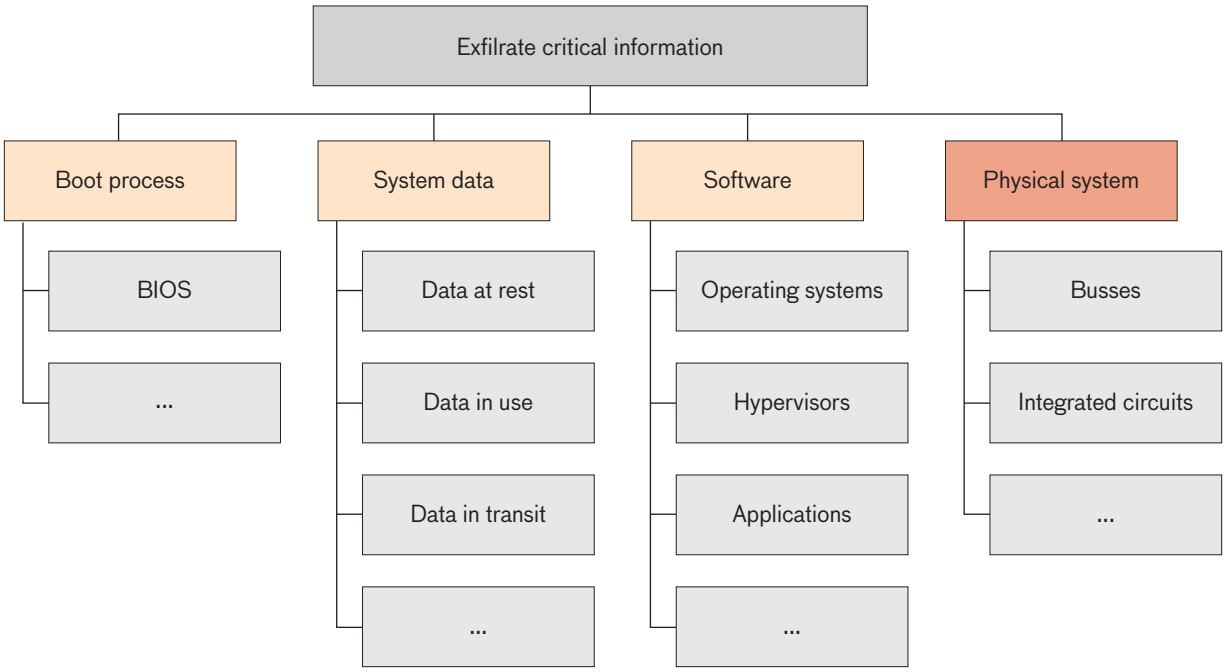

FIGURE 4. Example unmanned aircraft system (UAS) attack targets illustrate the vulnerabilities and sources of a threat scenario with three attack surfaces (boot process, system data, and software) and one physical attack surface (physical system).

Because the UAS is built with minimal system software for dedicated purposes, the exploitation of software vulnerabilities may be less likely than that for a general-purpose computer. The UAS has a strictly controlled provisioning environment accessible by a very limited number of authorized users, reducing the risk of introducing unverified and untrusted software into the UAS. However, one should always assume that an adversary will attempt to eavesdrop on wireless communication; thus, data protection is a high security priority.

Developers must also consider physical attacks because there is a high probability that adversaries will gain physical access to a UAS device, allowing enemies to reverse engineer the device or modify sensitive components in order to leapfrog their own technology or to gain unauthorized access to intellectual property. The most popular protection technique to date is the use of a strong protective enclosure equipped with electronic sensors to detect unauthorized accesses. However, because some systems are deployed and unattended for extended periods of time, it is challenging to maintain the standby power necessary for intrusion detection and response.

Developers must consider all threats and protect the confidentiality and integrity of the UAS data existing in three forms: data in use, data at rest, and data in transit. Various hardware and software solutions, most based on cryptography, are available à la carte. However, cryptographic technology must be fully integrated with the processor for efficient data protection via secure key management.

Security Metrics

Specifying and measuring security requirements for embedded system development are difficult. The requirements of the CIA triad for embedded systems are excellent objectives but are too abstract to be used as measurable security metrics to evaluate an embedded system during the design process. We have thus created three practical security metrics to facilitate the design of a secure embedded system: trustworthiness, protection, and usability. These metrics do not support absolute measurements but provide parameters to guide the design of embedded system security as the system's mission functionality architecture evolves. In addition, multiple system architectures can be qualitatively evaluated and

compared to determine relatively how well they provide security. Because these metrics are qualitative and subjective, each security decision must include sufficient justification and documentation. Developers evaluate each metric by analyzing system functionality and security against a specific CONOPS. For example, developers will measure trustworthiness and protection on the basis of the system's current defense mechanisms compared with the level of defense that the CONOPS requires. If the system is lacking in defense, it will be less trustworthy and unable to adequately protect information.

Trustworthiness is a qualitative analysis of the system's effectiveness in defending against potential threats relevant to its CONOPS. On the basis of current system design and system information fidelity, developers have a certain level of trust in system behavior during an attack. For example, if a system is equipped with a defense mechanism against a certain threat, the system's security and trustworthiness likely improve. While unpatched system vulnerabilities reduce security, understanding those vulnerabilities enables developers to add protection technology to the design.

The protection metric is a qualitative analysis of the system's capability to support added-in protection technologies and address vulnerabilities identified in a CONOPS. Together, the trustworthiness and protection metrics can be used to measure how well a system's security addresses confidentiality and integrity requirements.

Usability is a qualitative analysis of the system's suitability to a task. A system that is highly secure but incapable of delivering the required functionality is not designed well. Usability metrics evaluate a system's design by considering the system's throughput, resilience, portability, upgradability, SWaP, and other similar parameters.

A system's processing requirements, threats, and protection needs vary during the course of a system's operation. To evaluate a system during operation, we examine four phases:

1. Startup: The system is being booted into a state suitable for operations; a trusted computing base (TCB), the set of components that provide the system with a secure environment, is established.
2. Execution: The system is in the operational state and performs functions required by the mission.
3. Shutdown: The system is in the process of turning off.
4. Off: The system is powered down.

Architecture and Enabling Technologies

Because the critical information of a COTS-based embedded system is mostly in the system's software and firmware, cryptography is the foundation of the system's overall security. Many efficient, secure building blocks, such as the National Security Agency–approved Suite B cryptography [7], can be implemented with software, firmware, or hardware and are often obtainable as open-source intellectual property. However, simply using standard cryptographic primitives cannot guarantee the adequate implementation of security functions. Encryption effectiveness is based on the manner in which the cryptographic primitives (low-level cryptographic algorithms) are assembled and coordinated into the desired application-specific security functions. Encryption effectiveness also depends on key management, which includes the generation, distribution, and protection of keys.

Lincoln Laboratory has developed a solution to address encryption key management: Lincoln Open Cryptographic Key Management Architecture (LOCKMA), a highly portable, modular, open software library of key management and cryptographic algorithms that are suitable for embedded system uses. Designed to secure systems used in a wide range of missions, LOCKMA provides user, identity, and key management functions, as well as support for hardware and software cryptographic primitives, including the Suite B cryptographic primitives. LOCKMA has an intuitive front-end application programming interface (API) so developers can easily access LOCKMA's core functionality. To use LOCKMA, developers are not required to have advanced knowledge of the cryptography or key management algorithms implemented by LOCKMA's core modules; instead, they simply use the API to create security functions. LOCKMA handles the processing of key management messages and makes extensive use of cryptographic primitives available in several commercial and open-source libraries. Figure 5 shows LOCKMA's interfaces as high-level security functions and low-level cryptographic primitives.

Because software-implemented security functions may not meet extreme SWaP requirements, Lincoln Laboratory has implemented LOCKMA in a security coprocessor (S-COP), which applies cryptographic primitives in hardware. The benefits of hardware implementation over software implementation include much faster computation

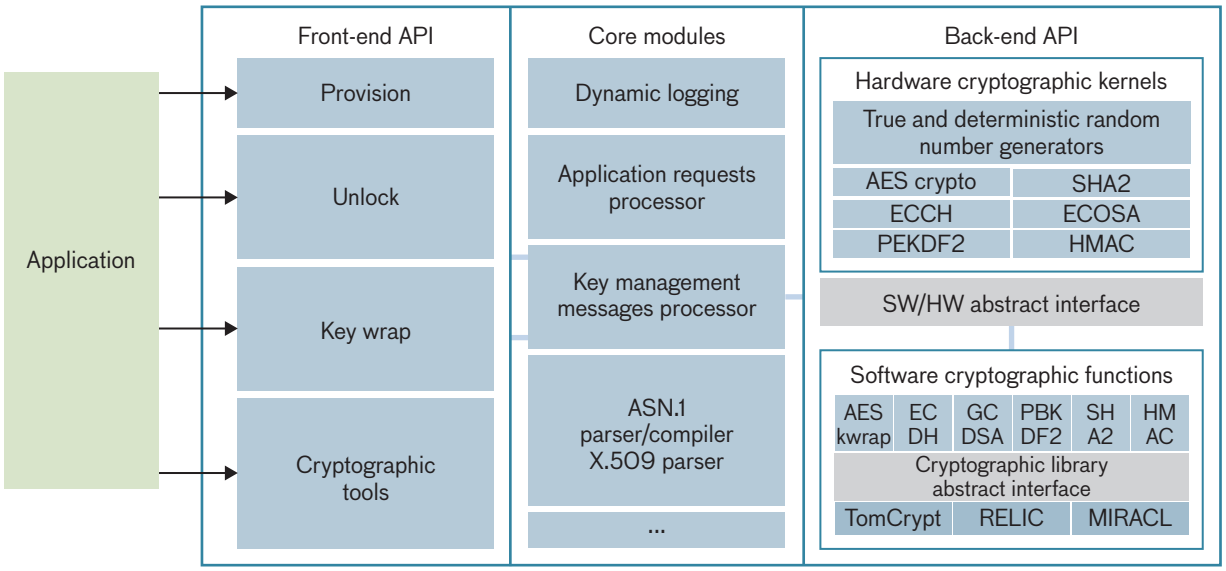

FIGURE 5. The LOCKMA software provides a front-end application programming interface (API) for high-level security functions that application developers can use directly. Complicated cryptographic algorithms are captured as core modules, which are hidden from application developers. The back-end API supports the use of low-level cryptographic kernels implemented in either hardware or software.

times, lower power consumption, hardware separation, and thus protection of sensitive keys from nonsensitive data and code.

Figure 6 shows the UAS embedded system architecture, previously shown in Figure 3, in which the CPU is secured with an S-COP and a physical unclonable function (PUF), which is a unique function that can be easily evaluated but hard to duplicate. The S-COP employs dynamic key management and accelerated Suite B cryptography for the authentication steps necessary to securely boot the CPU. The PUF provides an inviolable root of trust from which a unique cryptographic key is derived.

Lincoln Laboratory researchers have developed an optical PUF that can be implemented on a fully fabricated printed circuit board (PCB). As illustrated in Figure 7, the PUF is constructed by adding one or more light-emitting diodes (LED) and an imager to the PCB, which is then coated with a thin polymer planar waveguide. Upon powering up, the S-COP derives a unique numerical code from the imager, which receives light that is emitted by the LEDs and travels through the waveguide. This code is then used for device identification and key derivation. Manufacturing variations ensure a unique identification code for each PCB. Invasive attempts to learn about the PUF code (e.g., for

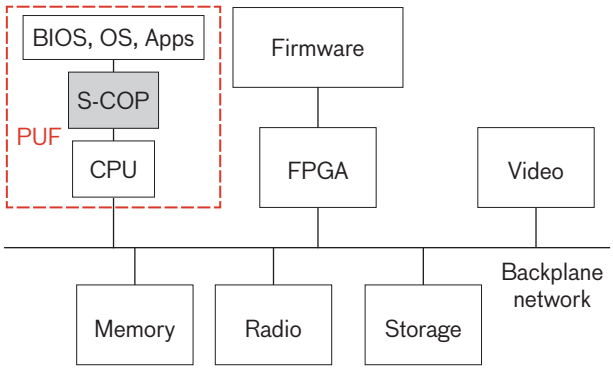

FIGURE 6. A security coprocessor (S-COP) is used along with a physical unclonable function (PUF) to secure a commercial off-the-shelf central processing unit (CPU).

cloning or other unauthorized actions), even when the PCB is unpowered, will disturb and damage the coating and irreversibly destroy the PUF code.

Because many environmental conditions, such as temperature and aging, can cause the PUF reading to vary, a technique called fuzzy extraction is employed to ensure that the same key will be derived from the PUF under various conditions [8]. This technique allows the S-COP to secure the boot process, load only trusted

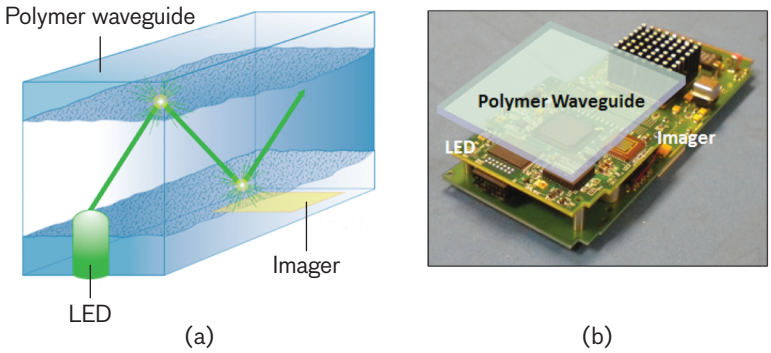

FIGURE 7. An optical physical unclonable function (PUF) is implemented with a waveguide. An operating concept illustration is shown in (a); implementation of the concept on a fully fabricated printed circuit board is shown in (b).

software, and confirm that the unique identity is intact before, during, and after the boot process. In addition to protecting data at rest with cryptography, the S-COP uses key management to support secure communications between subsystems to protect data in transit.

This S-COP–based secure embedded architecture allows software applications to be developed and tested initially without invoking security features. When a system is provisioned for deployment, developers apply the PUF to its PCB and load the finalized software code encrypted with the PUF-derived key. An incorrect PUF code will cause a failed software decryption, and the system will not start. The decoupling of the S-COP and the CPU allows DoD embedded systems to leverage mainstream CPUs, enhancing system usability and upgradability.

Figure 8 shows a test bed that we have developed to evaluate the S-COP–based secure architecture. In an unsecured architecture, the CPU reads in the basic input/output system (BIOS) and bootstraps the operating system (OS). Without authentication, the CPU is vulnerable to a maliciously modified BIOS and OS.

The S-COP–based secure architecture addresses this vulnerability by authenticating the BIOS, OS, and applications, as illustrated in Figure 9. When the embedded system powers up, the S-COP halts the CPU while the S-COP performs authentication. S-COP first reads the PUF and derives a key, which is used to decrypt the BIOS. If the decryption is successful, the CPU is released to execute the BIOS. The S-COP then authenticates and decrypts the OS and boots the system. Encrypted applications are loaded and handled in the same manner. In addition to associating an application with a designated system, the system can use LOCKMA key management to dynamically and seamlessly adjust the authorization of application execution (e.g., in time-specific and/or location-specific modes).

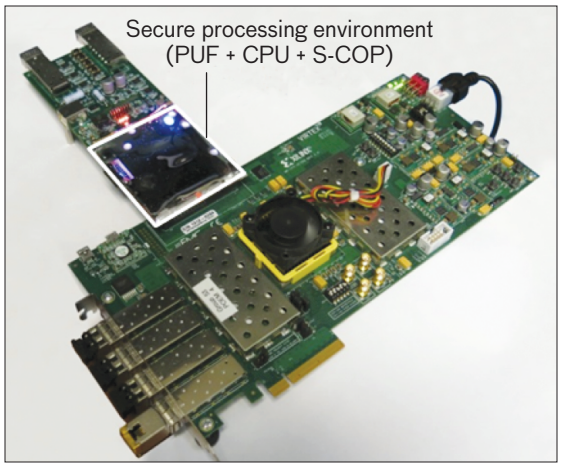

FIGURE 8. A secure processing environment integrates a central processing unit (CPU), a security coprocessor (S-COP), and a physical unclonable function (PUF).

Figure 10 shows data-at-rest and data-in-transit protection enabled by the S-COP. In System 1 and System 2, the S-COP encrypts the CPU-generated data before they are stored, thus protecting them from unauthorized access. Likewise, the S-COP decrypts stored data before sending them to the CPU. Figure 10 also shows the concept of using S-COPs to protect data in transit between two systems by establishing an encrypted communication channel over which encrypted data can flow.

Evaluation

In terms of the CIA triad, the S-COP addresses confidentiality and integrity by protecting the boot process, data, and communication channel from unauthorized access and alteration. The S-COP itself does not fully ensure a system's availability, but the decoupling of functionality and security, which allows for the use of a mainstream

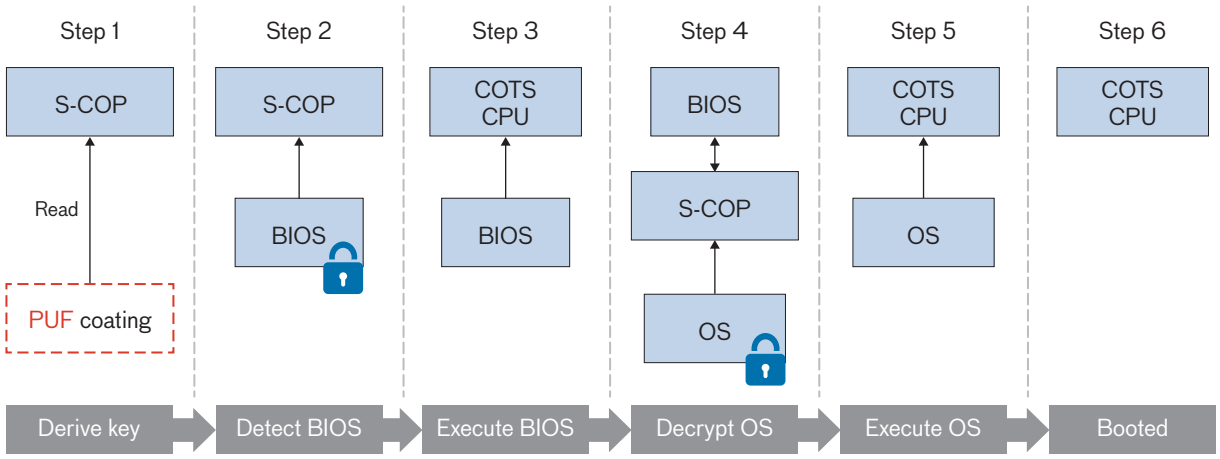

FIGURE 9. During the secure boot process, the central processing unit (CPU) is halted until the security coprocessor (S-COP) successfully verifies system integrity. Data that are protected by encryption are indicated by lock symbols.

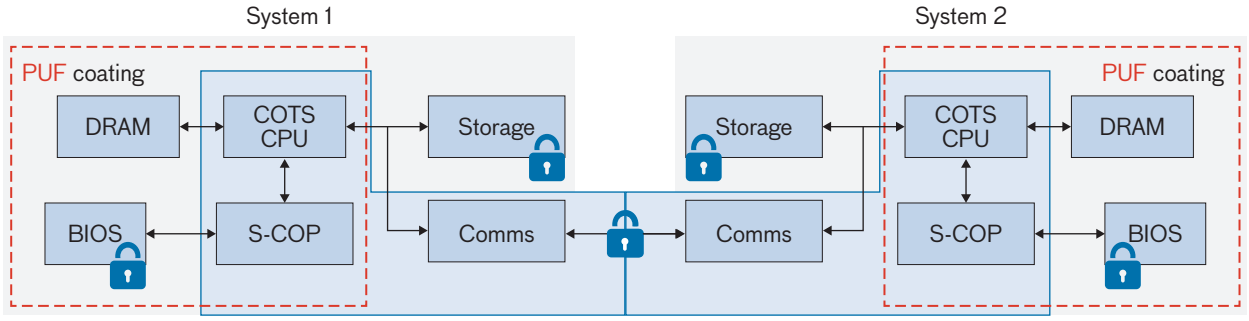

FIGURE 10. The security coprocessor (S-COP) enables data-at-rest and data-in-transit protection. Data that are protected by encryption are indicated by lock symbols.

CPU, results in improved system usability. The system can be adapted to support other agility and resilience measures, such as moving target technologies [9]. As an example, we evaluate the hypothetical UAS embedded system with an S-COP–based secure architecture by using the same three security metrics: trustworthiness, protection, and usability.

A mainstream unsecured CPU receives low trustworthiness ratings during all system operation phases, as we assume that it needs an inherently large trusted computing base (TCB) and lacks hardware-enforced boot attestation. The security of such a CPU enhanced with an S-COP dramatically increases across all system operational phases, earning the CPU increased trustworthiness ratings. However, during the execution phase, the user still needs to trust the OS, which may have inherent vulnerabilities. The trusted boot does not completely eliminate the risk of running untrusted or unverified codes that could potentially be exploited by attackers to escalate user privileges on the system or exfiltrate information.

If a CPU has no explicit support for physical protection, it will receive low protection ratings during the boot phase. Although the integration of a CPU with a TPM provides key storage and security measurements, the OS still needs to obtain, use, and revoke cryptographic keys, thus increasing the number of security components in the TCB. A lack of overall support for physical protection or for hardware-enforced encryption of code and data allows attackers to snoop or modify memory in the execution phase. During the off phase, the TPM could be physically replaced, and thus a new set of measurements could be inserted into the system. The S-COP–based secure architecture mitigates these deficiencies by creating a root of trust with a PUF and can be used to support physical protection.

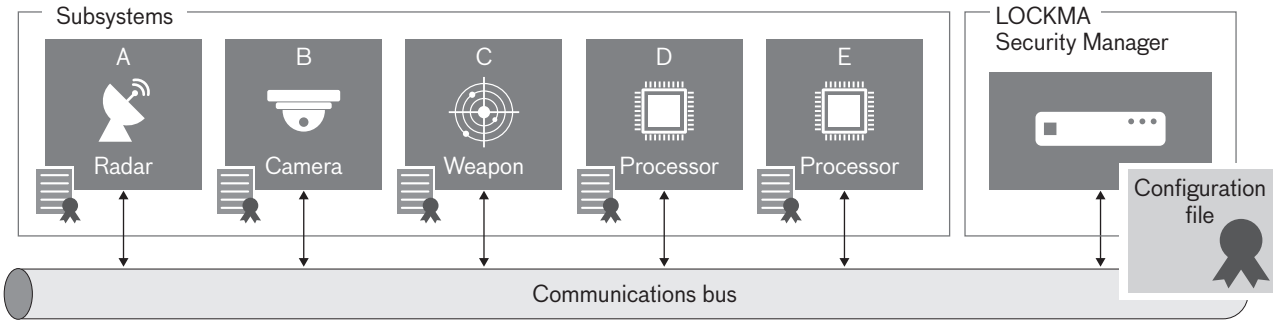

FIGURE 11. In a LOCKMA-based open-systems architecture security framework, the LOCKMA security manager (LSM) checks subsystem credentials against a config file to ensure that the configuration is authorized.

Because the S-COP can be adapted to secure a mainstream CPU, the usability of the secure UAS embedded architecture rates high; the architecture can leverage all the benefits of a COTS CPU, such as high performance (e.g., for signal processing), large cache and memory support, and widely supported software libraries.

Open-Systems Architecture Security

The use of OSAs can improve the development and life-cycle efficiency of system assets. Typically, OSAs incorporate several buses with well-defined interfaces for communication between components. A system can then be adapted to different needs by providing proper components and defining system interconnections.

Besides securing the CPU, LOCKMA is being developed into a cryptography-based secure framework that has been successfully demonstrated in OSA embedded system protection. The framework employs LOCKMA to provide encryption of data in use, data in transit, and data at rest to prevent eavesdropping, probing, and unauthorized access. In addition, developers can enforce a trusted configuration by accepting only predetermined payloads and preventing unauthorized hardware and/or software substitutes.

Figure 11 illustrates an example configuration that consists of several payloads and processors and a LOCKMA security manager (LSM). A digitally signed configuration (config) file that specifies authorized payloads, acceptable combinations of payloads, and secure communication channels establishes the authorized mission configuration. Figure 12 shows an example config file that has three sections: principals, constraints, and channels. The authorized subsystems are listed under the principals section; authorized configurations are noted under the constraints section; and authorized communication channels are specified in the channels section. In this example, the system can contain subsystems A, B, C, and D, among others. An authorized configuration is one that includes subsystem A or subsystem B with both subsystems A and D present. Subsystem A is given the role of a publisher (pub) and subsystems D and E are assigned the role of subscriber (sub). A digital signature is created for the config file so that its integrity can be verified.

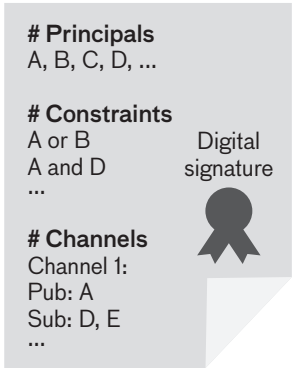

FIGURE 12. A security config file, an example of which is shown at left, is used to enforce payload authorization and secure communication channels.

At startup, the LSM verifies the digital signature of the config file and ensures that it is unaltered. Using the config file, the LSM collects subsystem credentials and confirms the absence of unexpected system payloads, leading to authorized system configuration. The system then starts and the LSM continues to set up secure communication channels.

Figure 13 illustrates how LOCKMA enables each subsystem with a key management (KM) function and an Advanced Encryption Standard (AES) encryption and decryption function. Subsystem A creates a key wrap containing a symmetric cryptographic key that is

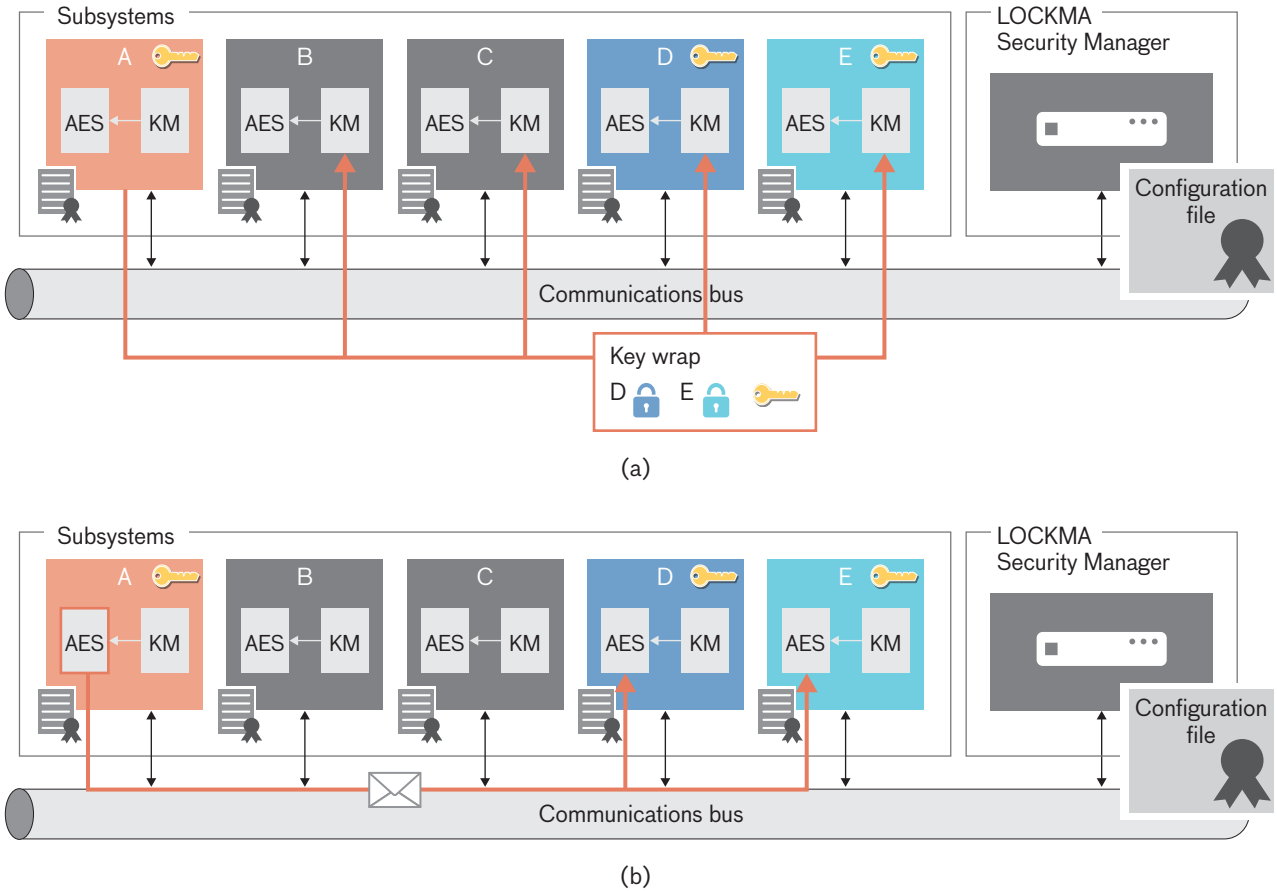

FIGURE 13. In a LOCKMA security framework, a publisher (e.g., subsystem A) sends a key wrap only accessible by intended subscribers (e.g., subsystems D and E) to retrieve a session key (a). The publisher and subscribers are then able to carry out encrypted communication (b). Each subsystem contains an Advanced Encryption Standard (AES) and a key management (KM) function.

only accessible by authorized subsystems D and E, and establishes a communication channel. The channel users retrieve the common secret session key and use it for encrypted communications. The system is then ready to perform its mission objectives.

Ongoing Work

Security has an asymmetric nature—an attacker can compromise a system by discovering a single, unexpected vulnerability, while a defender must defend against all vulnerabilities. Because it is impossible to correctly predict every future attack, securing an embedded system to prevent attacks is not a guarantee of mission assurance. Being secure is not adequate; systems must also be resilient. Lincoln Laboratory is vigorously pursuing an answer to the essential mission-assurance question: If an attacker is successful despite implemented defenses, what can be done so the mission can continue until completion?

Our objective is to define a standardized reference secure and resilient architecture for DoD embedded systems. We want to ensure that systems continue to function when a situation does not go as we expect. Our work is guided by the four stages of actions involved with the resiliency of an embedded system against cyber attacks: anticipate, withstand, recover, and evolve [10]. Our current research and development focuses on approaches that enable a system to defend against threats, withstand attacks and complete mission goals, recover from a degraded state and return to a normal state, and evolve to improve defense and resilience against further threats.

Our ongoing work also includes the development of mission-level resiliency metrics to answer the following question: Is the mission more likely to be successful

when our system is used? A system specification such as system restart time is a good design objective, but by itself does not provide information about system availability and mission assurance. We are developing a systematic approach to connect mission-level resiliency metrics to system specifications.

Acknowledgments

The authors would like to acknowledge technical contributions from the team members: Thomas Anderson, Walter Bastow, Robert Bond, Brendon Chetwynd, Robert Cunningham, Ford Ennis, Benjamin Fuller, Michael Geis, Karen Gettings, Antonio Godfrey, Raymond Govotski, Kyle Ingols, Eric Koziel, Joshua Kramer, Theodore Lyszczarz, and Merrielle Spain. ■

References

1. T.M. Takai, "Department of Defense Instruction, Subject: Cyber Security," 14 Mar. 2014, available at http://www.dtic.mil/whs/directives/corres/pdf/850001_2014.pdf.
2. D.R. Martinez, R.A. Bond, and M. Vai, eds., *High Performance Embedded Computing Handbook: A Systems Perspective.* Boca Raton: CRC Press, 2008.
3. Department of Defense Open Systems Architecture Data Rights Team, "DoD Open Systems Architecture Contract Guidebook for Program Managers V. 1.1," June 2013, available at http://www.acqnotes.com/Attachments/Open%20System%20Architecture%20(OSA)%20Contract%20Guidebook%20for%20Program%20Managers%20June%2013.pdf.
4. T. Huffmire, C. Irvine, T.D. Nguyen, T. Levin, R. Kastner, and T. Sherwood, *Handbook of FPGA Design Security*. New York: Springer, 2010.
5. Trusted Computing Group, "How to Use the TPM: A Guide to Hardware-Based Endpoint Security," 2009, available at http://www.trustedcomputinggroup.org/files/resource_files/8D42F8D4-1D09-3519-AD1FFF243B223D73/How_to_Use_TPM_Whitepaper_20090302_Final_3_.pdf.
6. Intel, "Software Guard Extensions Programming Reference," Sept. 2013, available at https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.
7. National Security Agency, "Suite B Cryptography," 25 Sept. 2014, available at https://www.nsa.gov/ia/programs/suiteb_cryptography/.
8. M. Spain, B. Fuller, K. Ingols, and R. Cunningham, "Robust Keys from Physical Unclonable Functions," *Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2014, pp. 88–92.
9. H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding Focus in the Blur of Moving-Target Techniques," *IEEE Security & Privacy*, vol. 12, no. 2, 2014, pp. 16–26.
10. D.J. Bodeau and R. Graubart, "Cyber Resiliency Engineering Framework," MITRE Technical Report, document number MTR110237, Sept. 2011, available at https://www.mitre.org/sites/default/files/pdf/11_4436.pdf.

About the Authors

Michael Vai is a senior staff member in the Secure Resilient Systems and Technology Group. From 2012 to 2015, he served as an assistant leader of this group. Previously, he was an assistant leader of the Embedded and Open Systems Group in the Intelligence, Surveillance, and Reconnaissance and Tactical Systems Division. He has worked in the area of embedded systems and technology for more than 25 years, leading the development of advanced embedded systems and publishing his research extensively. Prior to joining Lincoln Laboratory in 1999, he was on the faculty of the Department of Electrical and Computer Engineering at Northeastern University. During his tenure, he conducted multiple research programs funded by the National Science Foundation, Defense Advanced Research Projects Agency, and industry. His current research interests include secure and resilient embedded systems and technology, particularly systems involved in tactical operations. He received his master's and doctoral degrees from Michigan State University, in electrical engineering.

David J. Whelihan is a technical staff member in the Secure Resilient Systems and Technology Group. He started at Lincoln Laboratory in 2002 as a summer intern working on advanced radar processing hardware while earning his doctorate in electrical and computer engineering from Carnegie Mellon University. After graduating in 2004, he joined the startup company DAFCA, where he led the development of an advanced application-specific integrated circuit and field-programmable gate array debug tool. He then worked for a hardware cyber security startup. Prior to graduate school, he spent three years at Intel as an architectural validation engineer working on next-generation microprocessors. In 2010, he rejoined Lincoln Laboratory as a technical staff member, leading the Laboratory's efforts in the Defense Advanced Research Projects Agency's Photonically Optimized Embedded Microprocessors program and serving as the lead architect on the Self-Contained High-Assurance MicRO Crypto and Key-Management Processor system, which won a 2012 MIT Lincoln Laboratory Best Invention Award. His current work involves several programs dealing with secure and resilient hardware and software systems.

Benjamin R. Nahill is a technical staff member in the Secure Resilient Systems and Technology Group, specializing in secure hardware and embedded systems. He works on solving problems in secure processing and data protection by leveraging novel hardware architectures. Prior to joining the Laboratory in 2014, he worked in a variety of areas, including embedded system design in resource-constrained environments, satellite communications, and signal processing. He received a bachelor's degree in computer engineering and a master's degree in electrical engineering, both from McGill University.

Daniil M. Utin is a technical staff member in the Secure Resilient Systems and Technology Group. Previously, he cofounded several successful Internet technology and gaming-related companies, including WorldWinner.com and Cambridge Interactive Development Corporation. He has substantial expertise in developing secure, scalable, high-transaction-volume software systems coupled with dynamic, rich front-end graphical user interfaces. He joined Lincoln Laboratory in 2009 and has been designing cryptographic key management systems and utilizing his commercial software background to lead the development of secure usable solutions. He earned his bachelor's and master's degrees in computer science from Brandeis University.

Sean R. O'Melia is a technical staff member in the Secure Resilient Systems and Technology Group. He joined Lincoln Laboratory in 2008. His work to support secure communications in tactical networks includes the development of cryptographic key management technology for small unmanned aircraft systems. Most recently, he has concentrated on the design of a key management architecture for future tactical satellite communications capabilities and the development of security for open-architecture airborne platforms. He received bachelor's and master's degrees in computer engineering from the University of Massachusetts, Lowell. His graduate research focused on instruction set extensions for enhancing the performance of symmetric-key cryptographic algorithms.

Roger I. Khazan is the associate leader of the Secure Resilient Systems and Technology Group. He has led programs and conducted research in the areas of systems and communication security. His expertise and research interests are in usable security, key management and distribution, and applied cryptography, with specific focuses on disadvantaged tactical environments and embedded devices. He is passionate about creating technology that significantly simplifies the task of adding strong and usable cryptographic protections to software and hardware applications. He earned his master's and doctoral degrees from MIT in the Department of Electrical Engineering and Computer Science, and he completed a minor in management at the MIT Sloan School of Management. He earned a bachelor's degree from Brandeis University, with a double major in computer science and mathematics and a minor in economics.