

---

# The Seven Properties of Highly Secured Devices (2nd Edition)<sup>1,2</sup>

Galen Hunt, George Letey, and Edmund B. Nightingale

Microsoft Azure Sphere Team

---

## ABSTRACT

*Many organizations building and deploying IoT devices largely underestimate the critical societal need to embody the highest levels of cybersecurity in every network-connected device. Every connected child's toy, every household's connected appliances, and every industry's connected equipment needs to be secured against network-based attacks. Until now, high development and maintenance costs have limited strong security to high-cost or high-margin devices.*

*Our mission is to bring high-integrity cybersecurity to every IoT device. We are especially concerned with the tens of billions of devices powered by microcontrollers. This class of devices is particularly ill-prepared for the security challenges of internet connectivity. Insufficient investments in the security needs of these and other price-sensitive devices have left consumers, enterprises, and society critically exposed to device security and privacy failures.*

*This paper makes two contributions to the field of device security. First, we identify seven properties we assert are required for all highly secured devices. Second, we describe our experiment working with a silicon partner to create a prototype highly secured microcontroller, codenamed Sopris. We evaluate Sopris against the seven properties framework and in a penetration test utilizing a red team of 150 top-tier hackers. Our experimental results suggest that even the most price-sensitive devices can and should be redesigned to achieve the high levels of device security critical to society's safety.*

---

## 1. INTRODUCTION

The next decade promises the universal democratization of connectivity to every device. Significant drops in the cost of connectivity mean that every form of electrical device—every child's toy, every household's appliances, and every industry's equipment—will connect to the Internet. This Internet of Things (IoT) will drive huge economic efficiencies; it will enable countless innovations as digital transformation reaches across fields from childcare to eldercare, from hospitality to mining, from education to transportation. Although no person can foresee the full impact of universal device connectivity, anticipation of this new frontier is widespread [1] [2].

In our experience, the organizations and individuals building and deploying IoT devices largely underestimate the critical need for the highest levels of cybersecurity in every network-connected device. Even the most mundane device can become dangerous when compromised over the Internet: a toy can spy or deceive [3], an appliance can self-destruct or launch a denial of service [4], a piece of equipment can maim or destroy [5]. With risks to life, limb, reputation, and property so high, single-line-of-defense and second-best solutions are not enough. Because many connected devices will be deployed as components of larger IoT systems, the compromise of the even the most innocent device can easily lead to compounding risks through data pollution attacks, lateral movement attacks, or denial-of-service attacks.

---

<sup>1</sup> This edition adds results from the Sopris Security Challenge and improvements to the text from the [1<sup>st</sup> edition](#) based on feedback from the device security community. Key changes in the text are introduced with footnotes.

<sup>2</sup> We use the past-tense, "secured", to acknowledge that the challenges of securing a device require constant improvement; yesterday's security may be insecure today due to newly emerged security threats.

We don't intend to be alarmists. Although the state-of-the-art cybersecurity of internet-connected devices leaves much to be desired, we are quite optimistic for the future of device security. We believe it is within the realm of achievability for all devices, even the most price sensitive, to be engineered with sufficient security to merit trust even in the face of aggressive attacks from determined hackers.

Our fears and our hopes for connected device security are grounded in decades of Microsoft experience as an active defender in the Internet security battle. Early attacks against networked PCs motivated Microsoft to pioneer remote update of devices in the field with Windows 95 [6]. Escalating attacks motivated Microsoft to pioneer automated error reporting of security attacks and automated evidence analysis with Windows XP [7]. The desire to avoid in-field vulnerabilities continues to motivate Microsoft to create technologies and tools to detect and address vulnerabilities at design time [8] [9].

The goal of our work is to enable device manufacturers, regardless of industry, to incorporate the highest levels of cybersecurity in every network-connected device they build; and, by extension, to allow every consumer and organization to choose to deploy and use only devices with high levels of security. In our studies of existing devices and their cybersecurity capabilities, we have identified seven properties required for highly secured, network-connected devices: a hardware root of trust, defense in depth<sup>3</sup>, a small trusted computing base, dynamic compartments<sup>4</sup>, password-less authentication<sup>5</sup>, error reporting<sup>6</sup>, and renewable security (see Section 2).

For any network-connected device to remain secured, it must possess or be provided with all seven of these properties. To implement these seven properties, the hardware and software (firmware) of the device must work together, with device security rooted in hardware and guarded by continuously improving security software. Where one or more of the seven properties are not built into the device or have a substandard implementation, those properties must be augmented externally with additional human-based practices and processes—often at considerable expense. To give just one example, a device lacking fully automated renewable security must be manually updated—a practice that may require expensive truck rolls of technicians to deploy updates in the field in the event of the inevitable emergence of a significant new security threat. In our experience, this last point is critical because we find device manufacturers persistently underestimate the frequency of updates required by the emergence of new security threats—even the most well-designed devices typically require multiple updates per year.

We find these cybersecurity properties especially lacking in microcontroller-based devices. Some microcontroller families are beginning to evolve security features in hardware, such as cryptographic engines or trusted execution environments. However, these improvements don't go far enough. Providing cryptographic acceleration or private key storage isn't enough to create a highly secured device if the microcontroller doesn't also allow defense in depth or dynamic compartments. Most

---

<sup>3</sup> "Defense in depth" was reordered before "small trusted computing base" as we've found the latter often easier to explain after the former was defined. Likewise, the reordering of "renewable security" after "error reporting".

<sup>4</sup> "Compartmentalization" was changed to "dynamic compartments" to express the needed ability to change compartment boundaries over the lifetime of a device in the face of escalating security threats.

<sup>5</sup> "Certificate-based authentication" was changed to "password-less authentication" to remove the implied dependency on a specific implementation technology.

<sup>6</sup> "Failure reporting" was changed to "error reporting" to express that not all errors lead to failures.

microcontrollers, even newer microcontrollers with advanced security features, lack the capabilities needed for robust implementations of all seven properties of highly secured devices.

To address the cybersecurity challenges facing network-connected devices that are powered by microcontrollers, we enlisted the help of MediaTek to revise one of their existing microcontrollers, the MT7687, to create Sopris<sup>7</sup>, a proof-of-concept highly secured microcontroller (described in 3.1). Sopris is an experimental chip that allowed us to explore the ability to create experimental microcontroller-powered devices that embody complete implementations of the seven properties of highly secured devices. The key hardware innovations in Sopris are 1) the addition of a hardware root of trust embedded within a security processor in the microcontroller chip, and 2) the inclusion of a memory management unit (MMU) in the microcontroller's primary processor. These innovations create a microcontroller architecture that we believe will—with appropriate software—allow the creation of highly secured devices.

## 2. PROPERTIES OF HIGHLY SECURED DEVICES

Building secured devices is challenging. However, from observation of existing best-in-class devices, we argue security is more of a science than an art. If one adheres rigorously to well-understood principles and practices, from design through deployment, building secured devices is repeatable. Based on our observations, we have identified seven properties (see Table 1). We assert that these seven properties are required for all highly secured, network-connected devices.

**Highly secured devices have a *hardware root of trust*.** A device's private identity keys are protected by hardware, the integrity of device software is validated by hardware, and the hardware contains physical countermeasures against side-channel attacks. Unlike software, hardware has two important properties needed as foundation for device security. First, single-purpose hardware is resistant to reuse by an attacker for unintended actions. Second, hardware can detect and mitigate against physical attacks; for example, pulse testing the reset pin to prevent glitching attacks is easily implemented in hardware [10]. When used to protect secrets and integrity, hardware provides a solid root of trust upon which rich software functionality can be implemented securely and safely.

At a minimum, a hardware root of trust for IoT must provide at least two features: a secured device identity and software integrity protection through secure boot. A device-specific secret value stored privately within a device serves as a root for device identity. An orthogonal public key from the device manufacturer, stored securely within the device, can be used to authenticate the integrity of software intended to run on the device through signature verification. Better hardware roots of trust provide secured on-device generation of private keys, measurement of all software and firmware including ROMs, protection of data at rest, hardware write protection, trustworthy sources of entropy for random number generation, and irreversible storage for rollback protection.<sup>8</sup>

**Highly secured devices have *defense in depth*.** In highly secured devices, multiple mitigations are applied to each class of threat. In devices with only a single layer of defense, such as most RTOS-based devices, even a single error in design or implementation is sufficient to lead to catastrophic compromise.

---

<sup>7</sup> The Sopris name comes from the twin-summit Mount Sopris in the Elk Mountains of western Colorado.

<sup>8</sup> We have added examples for each property to help clarify the associated technical requirements.

Because new threats are often completely unanticipated, in practice, having multiple countermeasures often becomes the difference between a secured device and compromised device.

While minimum implementations of defense in depth help prevent device compromise, better implementations also provide mechanisms to reduce the impact of compromises. Both hardware and software should incorporate defense in depth. For example, write-once latches on configuration registers and write-protected latches on program code limit the extent to which a device can be repurposed even if the device’s software is temporarily compromised. Running critical tasks on physically isolated cores is another mechanism for providing defense in depth. As with hardware, defense-in-depth layers in the device’s software can prevent the exploit of a compromise in a networking stack to overwrite a device’s firmware in Flash storage.








Property	Examples and questions to test each property
 Hardware root of trust	Device identity and integrity are protected by hardware. Physical countermeasures resist side-channel attacks. <i>Does the device have a unique, unforgeable identity that is inseparable from the hardware? Is the integrity of the device software secured by hardware?</i>
 Defense in depth	Multiple mitigations applied against threats. Countermeasures mitigate the consequences of a successful attack on any one vector. <i>Does the device remain secured even if one security mechanism is breached?</i>
 Small trusted computing base	Private keys stored in a hardware-protected vault, inaccessible to software. Division of software into self-protecting layers. <i>Is the device’s security enforcement code protected from bugs in other software on the device?</i>
 Dynamic compartments	Hardware-enforced barriers between software components prevent a breach in one from propagating to others. <i>Is a failure in one component of the device contained to that component? Can new compartments be added in field to address new security threats?</i>
 Password-less authentication	Signed token, signed by an unforgeable cryptographic key, proves the device identity and authenticity. <i>Does the device authenticate itself with certificates or other tokens signed by the hardware root of trust?</i>
 Error reporting	A software error, such as a buffer overrun induced by an attacker probing security, is reported to cloud-based failure analysis system. <i>Does the device report errors for analysis to enable verification of the correctness of in-field device execution and identification of new threats?</i>
 Renewable security	Update brings the device forward to a secure state and revokes compromised assets for known vulnerabilities or security breaches. <i>Is the device’s software updated automatically? Can the device’s security TCB software be updated rapidly without repackaging other device code?</i>

Table 1. Required properties of highly secured devices with examples.

Common to almost all defense-in-depth designs are the philosophies of “assume breach” and “zero trust”—the designers evaluate effectiveness of each protection mechanism to guard security even if an attacker has already compromised other parts of the device. For example, defensive coding in the boot ROM of a secure chip might assume that an attacker has compromised the processor voltage and clock to produce “glitching” attacks against the ROM.

**Highly secured devices have a *small trusted computing base*.** A trusted computing base (TCB) is “a small amount of software and hardware that security depends on and that we distinguish from a much larger amount of software that can misbehave without affecting security” [11]. Within a device, the TCB for different operations may differ. For example, the TCB for securing data at rest may include the hardware root of trust, software for encryption and decryption, and software for sealing and unsealing crypto keys. On the other hand, the TCB for secure communication might also include an TLS implementation. The TCB for any operation should be kept as small as possible to minimize the surface that is exposed to attackers and to reduce the possibility that a bug or feature can be repurposed to circumvent security protections. The TCB code should be protected from non-critical device code to ensure its correct operation even if the other code is compromised. Less secured devices often have no isolated TCB—security code in these devices executes in the same compartment as the rest of the device code with the result that just one bug, anywhere in the device’s code, can lead to a catastrophic full-system compromise.

In minimum implementations, critical cryptographic key operations are isolated to a small TCB. For example, a device’s private identity key and access to it should be limited to the smallest possible subset of the device’s hardware and software. In better implementations, the TCB is layered to protect access to persistent storage, to protect access to critical I/O resources, to detect and recover from compromises in code above the TCB, and to fail over to protected backup software in the case of catastrophic device compromise. Code within the TCB should be rigorously tested for potential compromises by skilled experts well versed in the latest tools and methods employed by attackers.

**Highly secured devices provide *dynamic compartments*.** In a computing device, cybersecurity compartments are hardware-enforced boundaries that prevent a breach or flaw in one software compartment from propagating to other software compartments of the device. Compartments introduce additional protection boundaries to create additional layers of defense in depth. Dynamic compartments allow the introduction of new boundaries, throughout a device’s deployed lifetime, as required to improve security against escalating security threats.

Minimum implementations of dynamic compartments use operating systems processes or independent virtual machines as compartments. Devices with dynamic compartments can be updated in the field to incorporate significant security improvements to counter newly emerged security threats and attacks. Less secured devices, including most low-cost devices employing a real-time operating system (RTOS) design, have either no software compartments or only a small number of fixed compartments that cannot be reconfigured after a device is deployed and thus have very limited ability to evolve to address new security threats.

**Highly secured devices use *password-less authentication*.** Password-less authentication, such as certificates, are used to prove identities for mutual authentication when communicating with other local devices and with cloud services. A certificate or other password-less authentication token is a proof of identity and authorization that is signed with a secret private key and can be validated against a known public key. Unlike passwords or other authentication mechanisms that are based on shared secrets, password-less authentication mechanisms, backed by a hardware root of trust, can’t be stolen, forged, or otherwise used to authenticate an impostor.

In minimum implementations, certificates attest to not just the identity of the hardware but also the identity of the software running on the device—for example certifying that the device is running the latest software, which addresses all known security vulnerabilities. When a certificate or authentication token is issued based on attestation, services accessed by a device can verify and require that the device

perform a software update before proceeding if it is running an older version of its software with a known compromise.

**Highly secured devices have online *error reporting*.** When an error occurs on a secured device, an error report is collected automatically and sent to an error analysis system in a timely manner. In the best case, the error was caused by inadequate programming for an extremely rare sequence of events. In the worst case, the error was caused by an attacker probing the device for new attack vectors. Whichever the case, an error analysis system correlates error reports across an entire fleet of devices to allow automated diagnosis of errors. With a sufficiently large reporting base, even extremely rare error conditions can be diagnosed and corrected, and new attack vectors can be identified and isolated before they are widely exploited [7]. Error reporting enables a global “immune system” across a fleet of highly secured devices. Without automated online error reporting, device manufacturers are left in the dark as to the device errors experienced in the field and may be caught off guard by emerging attacks.

In minimum implementations, information about errors detected in hardware and in software are collected and aggregated to the device manufacturer. Errors are analyzed in near real time and a device manufacturer—or its software providers—routinely employ data-driven practices to diagnose and to correct errors encountered in the field through software updates.

**Highly secured devices have *renewable security*.** A device with renewable security can update to a more secure state automatically, even after the device has been compromised. Renewable security is necessary because security threats evolve and escalate as attackers discover new attack vectors and create new attack methods and tools. To counter emerging threats, device security must be renewed regularly. In extreme cases, when compartments and layers of a device’s software are compromised by zero-day exploits, lower layers must rebuild and renew the security of higher levels of the device. Remote attestation and rollback protections guarantee that, once renewed, a device cannot be reverted to a known vulnerable state. A device without renewable security is a crisis waiting to happen.

In minimum implementations, each layer of a device’s software defenses can be independently and automatically updated without invalidating the other layers. For example, an error in a networking protocol—such as the KRACK vulnerability discovered in the WPA2 WiFi protocols in 2017 [12]—can be updated without updating or retesting a device’s real-time control loops. Automatic update means that no manual intervention is required at the device to initiate an update. In addition to layer-independence and automation, reliable implementations of renewable security provide robust mechanisms to automatically recover even from failed updates or extraneous conditions such as power or network failures during an update.

### **3. SOPRIS: A PROTOTYPE HIGHLY SECURED MICROCONTROLLER**

The motivating hypothesis of our work is that even the most price-sensitive devices can be redesigned to become highly secure. Since the clear majority of the world’s devices<sup>9</sup> are driven by microcontrollers, the clearest test of our hypothesis is to build a microcontroller-based device that can meet all seven properties required of highly secured devices. To validate our hypothesis, we created a prototype secured microcontroller, codenamed Sopsis, and a secured operating system to support it. We created a number of devices based on Sopsis and then evaluated the security of the Sopsis-based devices against

---

<sup>9</sup> According to Databeans over 9 billion new microcontroller-based devices are built and deployed each year [16].

the seven properties framework by recruiting a red team of 150 top-tier security researchers for penetration testing.

### **3.1. CREATING A SECURED ROOT OF TRUST**

We set out to prove that practically any type of device can be converted into a highly secured device with a set of modest changes. We hypothesize that this can be done by 1) including within the device's primary processing chip an isolated security processor to provide a hardware root of trust, and 2) modifying the rest of the device hardware architecture to support defense in depth and compartmentalization mechanisms.

For a hardware root of trust, we integrated the Microsoft Pluton security processor originally developed and perfected in Microsoft's Xbox One gaming consoles.<sup>10</sup> Integrating Pluton, or an equivalently featured embedded security processor, into primary processing chip of the device is a significant necessary step towards creating highly secured devices and a significant improvement over off-chip security designs such as TPMs [13]. TPMs are connected to the CPU using a communication channel, typically a bus interface. The problem with using the bus interface as the primary communication channel is that it can be compromised using a physical attack [14]. The Pluton design builds security directly into the chip, completely removing the potential for that communication channel to be attacked.

The Pluton security processor is built into the device's microcontroller or other primary processing chip. It includes a dedicated CPU, cryptographic engines, a hardware random number generator (RNG), a key store, and a cryptographic operation engine (COE). The cryptographic engines in Pluton include an AES [15] symmetric-key decryption and encryption engine, a SHA [16] hashing engine for measuring code and checking certificates, and a public key engine for accelerating RSA [17] and ECC [18] public key operations. The hardware RNG is used to randomize the execution of the boot firmware so an adversary can't precisely time an attack, and for key generation and other cryptographic needs. The COE performs operations that require more than one cryptographic engine. An example of a COE command is a device attestation operation where the SHA engine is used to append a code measurement register to a challenge nonce from a certificate authority and then the result is signed using the device's private attestation key—all performed entirely in hardware so that no software has access to the device's attestation key.

### **3.2. CREATING A SECURED MICROCONTROLLER**

To create Sopor, we started from an existing state-of-the-art microcontroller, the Wi-Fi-enabled MT7687 [19] from our silicon partner MediaTek. The original MT7687 has a 192MHz ARM Cortex-M4 CPU, 352KB RAM, 28 GPIO pins, a 12-bit ADC, a complete Wi-Fi subsystem including an Wi-Fi 4 (802.11n) radio (both baseband and MAC), and a controller for the in-package Flash die (see Figure 1).

With MediaTek's assistance we modified and extended the MT7687. We made four changes to the MT7687 to convert it into Sopor (see Figure 2): we added the Pluton security processor to provide a hardware root of trust, we upgraded the primary CPU from an ARM Cortex-M4 to an ARM Cortex-A7 in order to include a memory management unit (MMU), and we increased the amount of on-die SRAM

---

<sup>10</sup> A pluton is a geographic formation resulting from a mass of magma slowly cooling beneath the earth to form the heart of a mountain, such as Mt. Sopor.

from 352KB to 6MB and the amount of second-die Flash from 2MB to 16MB to support a wide range of experiments.

From a security perspective, the most important changes were the addition of the Pluton security processor and the addition of address translation in the CPU through an MMU. The Pluton security processor forms the hardware root of trust for Sopsris—including controlling the entire boot sequence. Unlike the much more primitive memory protection unit (MPU) found in the Cortex-M4 and similar MPUs found in most microcontrollers, the MMU on the Sopsris Cortex-A7 processor enables multiple layers of isolation and multiple independent address spaces from which an OS can create process-isolation compartments. The MMU is used for fine-grained access control and address translation, but not for paging as on embedded MPUs and larger SOCs. Contrary to common expectations, our upgrade to the Cortex-A7 processor had minimal impact on die area and chip cost because of optimizations enabled by synthesizing the core for a low clock speed. The increase in on-die SRAM allows easy experimentation with many OS configurations while maintaining the security of on-die memory.

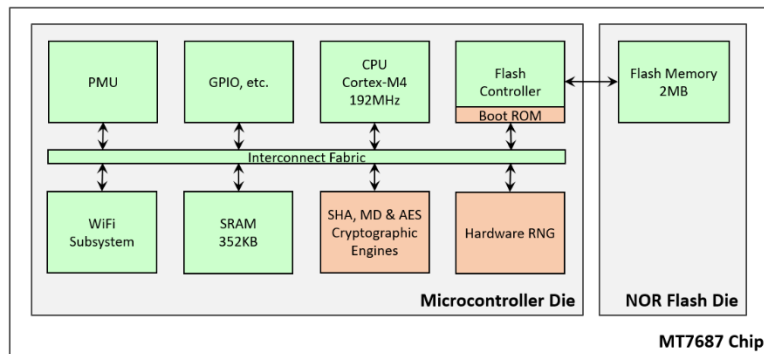


Figure 1. Architecture of the MT7687 Wi-Fi-enabled microcontroller.

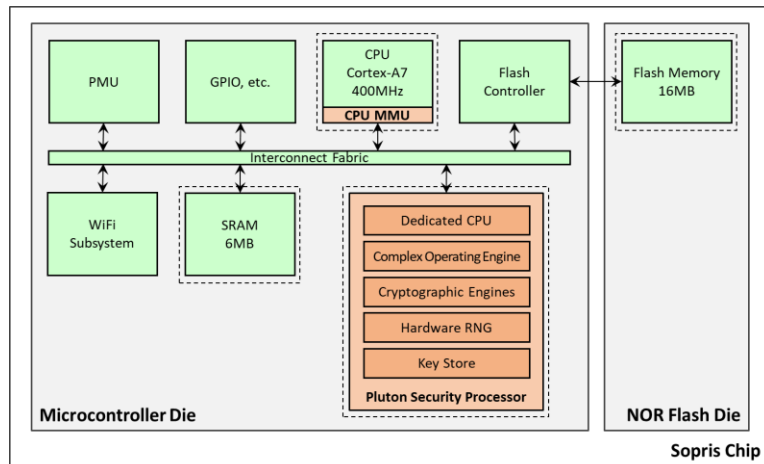


Figure 2. Architecture of the Sopsris highly secured, WiFi-enabled microcontroller.

We have incorporated MediaTek’s Sopsris prototype microcontroller into a small number of prototype devices. Figure 3 shows a prototype USB-powered developer board based on Sopsris.



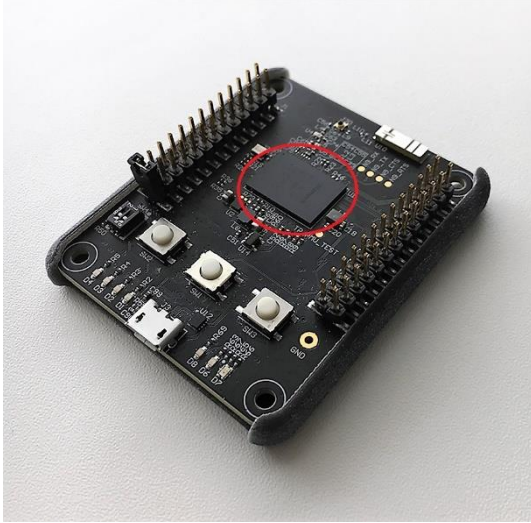


Figure 3. A Sopris developer board. Sopris is the black rectangular chip circled in red.

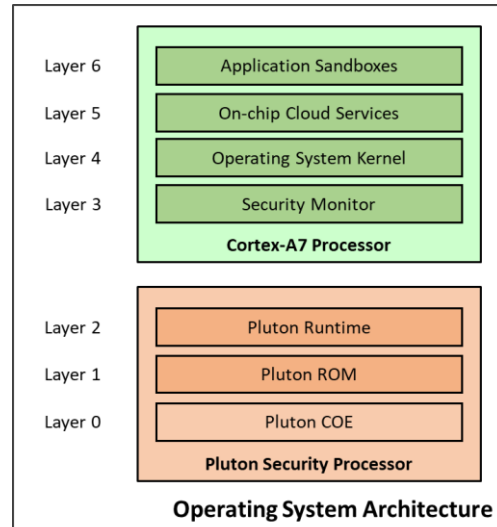


Figure 4. Architecture of our prototype operating system; Layers 0-2 execute in the Pluton security processor, Layers 3-6 in the primary processor.

### 3.3. CREATING A SECURED OPERATING SYSTEM<sup>11</sup>

Implementing the seven properties of a highly secured device requires both hardware and software. Figure 4 illustrates the multi-layered architecture of the prototype operating system powering our Sopris device. Layers 0 through 2 execute within the Pluton security processor. Layer 0, the Pluton COE, is implemented entirely in hardware state machines; Layers 1 and 2, the Pluton boot ROM and Pluton Runtime, execute on the Security Processor CPU from ROM and Flash respectively. Layers 3 through 6 and execute on the chip’s Cortex-A7 processor from Flash; Layer 3, the Security Monitor, executes in the TrustZone secure world environment; Layer 4, the operating system kernel, executes in Supervisor mode; and Layers 5 and 6, the on-chip cloud services and application sandboxes, execute in isolated address spaces in User mode.

Our prototype OS employs defense-in-depth mechanisms both between and within layers of the OS. Between layers of the OS, lower layers are designed on principles of zero trust—each layer assumes that the layers above it have been completely breached by an attacker—so requests and data from higher layers are not trusted until they have been fully validated. For example, write access to persistent Flash storage is strictly controlled by the Security Monitor and Pluton Runtimes, which prevent even a compromised OS kernel from overwriting application or kernel images in Flash storage.

Within each OS layer, available hardware protection mechanisms are used to provide additional protection beyond that offered by best-practice defensive coding techniques. For example, the Pluton ROM and Pluton Runtime both take advantage of the MPU mechanisms in the Security Processor CPU to reduce the scope of attack of potential data processing vulnerabilities. Similarly, the layers of the OS running on the Cortex-A7 take advantage of its MMU mechanisms to further reduce the scope of attack for potential data processing vulnerabilities and ROP attacks using MMU-enabled defense-in-depth techniques such as sparse address spaces, guard pages, and address space layout randomization.

<sup>11</sup> This description of our prototype OS is new to this second edition of the paper.

### 3.4. PROPERTY-BASED EVALUATION

Table 2 contrasts the highly secured device properties found in Sopris with the properties found in traditional microcontrollers, such as the MT7687 from which it was derived. Sopris supports all the properties required to create a highly secured device as identified previously in Section 2.

Property	Supported by traditional MCU	Supported by Sopris
Hardware root of trust	No	Yes
Defense in depth	No	Yes
Small trusted computing base	No	Yes
Dynamic compartments	No	Yes
Password-less authentication	No	Yes
Error reporting	No	Yes
Renewable security	No	Yes

*Table 2. Properties of highly secured devices supported by a state-of-the-art traditional microcontroller and our experimental Sopris chip.*

The Sopris proof-of-concept provides an in-chip hardware root of trust—device secrets and software integrity are protected by the Pluton security processor. Sopris provides a small trusted computing base—for many operations, the TCB for Sopris is isolated to the processor within the Pluton security processor. Sopris supports defense in depth: between the MMU-enhanced CPU and the Pluton security system, up to seven layers of defenses have been implemented on Sopris. Sopris supports dynamic compartmentalization: separate compartments are implemented using isolated address spaces enabled by the MMU and new compartments can be introduced through software updates. Sopris supports hardware-protected password-less authentication: private keys stored in the Pluton security processor can form the basis of a secure per-device certificate chain protected from identity theft even of a compromised OS kernel. Sopris supports online error reporting: exception handling support in Sopris processors all collection of data about errors and for relay of that information to an error analysis service through Wi-Fi. Finally, Sopris supports renewable security: the multiple layers of protection enabled by Sopris can be used to provide ongoing security improvements in the face of emerging security threats without requiring revalidation and recertification of higher layers of the software stack.

In comparison to Sopris, existing microcontrollers do not support the properties required to create highly secured devices. For example, even though a microcontroller might include hardware crypto engines, keys controlled by software in the microcontroller’s primary processor are insecure if a flaw is found in that software. While we have used the architecture of Sopris to implement seven layers of defenses in the OS architecture, in contrast, traditional microcontrollers usually support at most two or three layers. In practice almost all RTOS-based devices use only one layer of defense. In such devices, virtually any software vulnerability can become a fatal flaw that allows an attacker to breach the single security layer and gain complete and permanent control of the device by reprogramming its Flash storage.

The positive impact on device cybersecurity of the MMU in Sopris cannot be overstated. The greatest weakness of the MPUs in traditional microcontrollers is that they lack address translation, meaning the addresses of code and data, both SRAM and Flash, can’t be remapped. As result, an MPU cannot implement known best practices for defense in depth such as sparse address spaces, guard pages, and

address space layout randomization. While it was quite simple for our prototype OS to support independent servicing of each layer of the OS and the application, independent servicing is, in practice, never implemented in an MPU-based microcontroller RTOS. This failure to provide independent services is because the RTOS and non-TCB components of the device firmware are almost always linked together in a single binary image and because independent servicing cannot be achieved efficiently without address translation.

While traditional microcontrollers are ill-fitted to the task of creating highly secured devices, Sopris proves that it is possible to construct a microcontroller that can readily provide the basis for highly secured devices.

### 3.5. RED TEAM PENETRATION TESTING EVALUATION<sup>12</sup>

While the seven properties speak to the potential of a device to be highly secured, additional testing is required to validate that the implementation of those properties is actually effective in repelling and countering attacks. For example, a device might have a small TCB, but there may be vulnerabilities within the implementation of the TCB. In practice, one of the most effective methods to evaluate the security of a device is through penetration testing by so-called “red team” of attackers. In these exercises, a red team of experienced hackers is recruited to attack the device with the latest tools and techniques that an adversary might use.

To evaluate the effectiveness of the security implemented our Sopris prototype, we conducted a red team security challenge comprising 150 top-tier security researchers. Our Sopris Security Challenge was administered by a third party, HackerOne. From its community of over 100,000 hackers, HackerOne curated a set of 150 top-tier researchers to focus on identifying vulnerabilities in Sopris. Each hacker received a developer board and documentation. Hackers were given 60 days to attack their Sopris device. To incentivize participation and effort, hackers were offered a range of bounties from a total reward pool of \$100,000 for breaching Sopris security. Individual bounties ranged from \$2,500 to \$15,000 based on their severity and reach (see Table 3).

Vulnerability reach	Privilege escalation	Information disclosure	Remote code execution
Users space	N/A	\$2,500	\$5,000
Kernel: Normal World	\$2,500	\$5,000	\$10,000
Kernel: Secure World	\$5,000	\$10,000	\$15,000
Secure silicon	\$10,000	\$15,000	\$15,000

*Table 3. List of bounties offered to hackers participating in the Sopris Security Challenge.*

Identified vulnerabilities were submitted to HackerOne for verification. HackerOne’s security analysts triaged and classified all identified vulnerabilities against the CWE standard [20] and assigned a severity ranking based on the CVSS v3.0 standard [21]. HackerOne distributed bounties directly to hackers after each vulnerability had been verified by their security analysts.

Hackers participating in the Sopris Security Challenge were rewarded only for attacks through the device’s network interface. Parallel to the security challenge, our internal red team ran several physical

---

<sup>12</sup> This section, documenting the results of the Sopris Security Challenge, was added to the second edition of this paper. The Sopris Security Challenge had not completed at the time of publication of the first edition.

cybersecurity attacks against Sopris chips. For example, in one attack, a Sopris chip was acid washed and examined with an electron microscope in an unsuccessful attempt to recover the device-specific private key from the chip. We believe this test is relevant because highly proficient attackers, such as organized crime or nation-state sponsored groups, would have the resources to successfully extract a key in this manner. It is important to note that such an attack would be of only a limited value as it would extract the private key for only a single chip and would simultaneously destroy the chip.

Sopris was resilient to all attacks created by the 150 security researchers during the 60-day challenge. Only two bugs, both information level, emerged from the Sopris Security Challenge. Neither of them resulted in an information disclosure, a remote code execution, or a privilege escalation. In hindsight, one drawback of the Sopris Security Challenge was that hackers were not given source to software on the device. While experience teaches that the lack of source code is not a long-term obstacle for hackers, we believe that providing source code would increase the thoroughness of the penetration testing performed by the hackers during a time-bounded experiment.

The results of the red team evaluation provide strong validation that the seven properties, when implemented well, are sufficient to ensure resilient security. The red team evaluation further demonstrates that the seven properties can be implemented effectively in even a microcontroller-class device such as Sopris. Tight budgets should be no excuse for substandard security.

#### **4. FROM SOPRIS TO AZURE SPHERE**

Our early experiments with Sopris led to the creation of Azure Sphere. Azure Sphere is an end-to-end solution designed to enable any device manufacture to create highly secured devices. Azure Sphere consists of three technical components: chips, an operating system, and a cloud service. Azure Sphere chips build upon the Pluton security processor design of Sopris, adding additional hardware security features such as stick bits to prevent compromise through unexpected elevations of privilege within software and in-fabric communications firewalls to create additional layers of defense within the hardware. The in-fabric communications firewalls enable Azure Sphere chips to support safety-critical real-time computation and highly secured network connectivity. The Azure Sphere operating builds on the architecture design we prototyped in Sopris to add fully highly robust, fully automated and fully independent update of each layer the OS. The Azure Sphere Security Service (AS3) works with the chip and OS to provide the cloud-related properties of a high secured device—password-less authentication, online error reporting, and cloud-powered renewable security.

More than meeting the security baseline of the seven properties, Azure Sphere is designed to allow any device manufacture—regardless of their level of expertise in security—to build highly secured devices. Firstly, Azure Sphere provides best-in-class implementations of all seven properties [22]. Secondly, Azure Sphere is offered not as a set of building blocks requiring significant assembly—as is common with an RTOS—but as a fully assembled solution operated by Microsoft as a service. In the event of a major internet security incident, such as the discovery of a vulnerability in a widely used networking prototype [12], a device manufacturer using Azure Sphere does not need to wait for an RTOS patch, recompile their firmware with the new RTOS patch, retest and revalidate their firmware, distribute the new firmware to their customers, or deploy technicians to ensure that devices deployed globally are updated with the new firmware—all as quickly as possible to in the case of zero-day attack. Instead, the device manufacturer using Azure Sphere need do nothing more than watch as the Azure Sphere team creates

and deploys an update to all the manufacturer's devices globally. No recertification of manufacturer code, even safety-critical real-time code, is required because Azure Sphere's defense-in-depth mechanisms and implementation of renewable security allow the TCB, network security code, and all other OS components to be updated to improve device security independently.

## **5. CONCLUSION AND CALL TO ACTION**

The coming decade will likely see the deployment of billions upon billions of network-connected devices. Although we applaud those consumers and organizations who have begun to recognize the critical importance of security in these coming devices, we see evidence that many stakeholders fail to appreciate the need to require that every IoT device have the high levels of security.

Our desire to bring the highest level of security to even the most commonplace of devices has led us to think deeply about the properties of highly secured devices. We have identified seven properties we assert are critical in all network-connected devices: a hardware root of trust, defense in depth, a small trusted computing base, dynamic compartments, password-less authentication, error reporting, and renewable security.

Grounded in the understanding of these seven properties of highly secured devices, we have set out to demonstrate that it is possible to bring these properties to low-cost applications. Our first milestone has been a step in that direction based on Sopris: building a test device that utilizes a prototype microcontroller and an operating system with these properties. Using property-based and red team evaluations (Sections 3.4 and 3.5), we have demonstrated through proof-by-existence that even single-chip connected devices can provide high levels of security. We have further demonstrated that these devices can prove resilient in the face of even persistent professional security hackers, provided that the device has a robust implementation of all seven properties. Subsequent to Sopris, our team has developed Azure Sphere, an end-to-end solution that allows any device manufacture to create highly secured devices.

Based on our preliminary experimental experience, we are confident that any IoT device can be designed to achieve high levels of device security—levels that will be critical to society's safety in the near future.

We call on device manufacturers, and on consumers and organizations deploying IoT devices, to accept nothing less than highly secured devices—devices that incorporate all seven properties of highly secure devices.

## **ACKNOWLEDGEMENTS**

The Sopris experiments built heavily on the work of others and we are grateful for their long hours and dedication from which we have benefited. We owe a special debt of gratitude to George Jen and team at MediaTek, who provided the original MT7687 design and worked with us to create the Sopris chip. We are grateful to our many colleagues on the 4x4 and Microsoft Azure Sphere teams, many of whom provided valuable feedback on drafts of this paper, and who built the prototype Sopris operating system and the Azure Sphere solution. We are inspired by their commitment to our mission to empower every organization on the planet to connect and create secured and trustworthy IoT devices.

## REFERENCES

- [1] C. MacGillivray and A. Wright, "Worldwide Internet of Things Connectivity Forecast, 2017–2021," IDC, 2017.
- [2] D. W. Cearley, B. Burke and M. J. Walker, "Top 10 Strategic Technology Trends for 2016," Gartner, 2016.
- [3] C. Wiking, "If Your Child Has This Doll You Should Get Rid of It Now," 17 Feb. 2017. [Online]. Available: <https://mom.me/news/39826-if-your-child-has-doll-you-might-want-destroy-it/>. [Accessed 17 Feb. 2017].
- [4] N. Perlroth, "Hackers Used New Weapons to Disrupt Major Websites Across U.S.," New York Times, 21 Oct. 2016.
- [5] E. Mills, "Internet-Connected Coffee Maker Has Security Holes," CNET, 17 Jun. 2008.
- [6] C. Gkantsidis, T. Karagiannis, P. Rodriguez and M. Vojnovic, "Planet Scale Software Updates," in ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pisa, Italy, 2006.
- [7] K. Kinshuman, K. Glerum, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle and G. Hunt, "Debugging in the (Very) Large: Ten Years of Implementation and Experience," Communications of the ACM, vol. 54, no. 7, pp. 111-116, 2011.
- [8] E. Bounimova, P. Godefroid and D. Molnar, "Billions and Billions of Constraints: Whitebox Fuzz Testing in Production," in International Conference on Software Engineering, San Francisco, CA, 2013.
- [9] P. Godefroid, P. de Halleux, M. Y. Levin, A. V. Nori, S. K. Rajamani, W. Schulte and N. Tillmann, "Automating Software Testing Using Program Analysis," *IEEE Software*, vol. 25, no. 5, pp. 30-37, 2008.
- [10] J. Boone and I. Zhuravlev, "There's A Hole In Your SoC: Glitching The MediaTek [MT8163V] BootROM," 15 October 2020. [Online]. Available: <https://research.nccgroup.com/2020/10/15/theres-a-hole-in-your-soc-glitching-the-mediatek-bootrom/>. [Accessed 15 October 2020].
- [11] B. Lampson, M. Abadi and M. Burrows, "Authentication in Distributed Systems: Theory and Practice," *ACM Transactions on Computer Systems*, 1992.
- [12] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, Dallas, TX, 2017.
- [13] International Organization for Standardization (ISO), "ISO/IEC 11889-1:2009 - Information Technology - Trusted Platform Module".
- [14] D. Andzakovic, "Extracting BitLocker Keys from a TPM," 13 March 2019. [Online]. Available: <https://pulsesecurity.co.nz/articles/TPM-sniffing>. [Accessed 15 October 2020].
- [15] National Institute of Standards and Technology, "197, Advanced Encryption Standard (AES)," *Federal Information Processing Standards (FIPS)*, 2001.

- [16] National Institute of Standards and Technology, "180-4, Secure Hash Standard," *Federal Information Processing Standard (FIPS)*, 2012.
- [17] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Strong Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1977.
- [18] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the Theory and Application of Cryptographic Techniques*, Springer, Berlin, Heidelberg, Germany, 1985.
- [19] MediaTek, Inc., "MT7687F Datasheet," Hsinchu, Taiwan, 2016.
- [20] MITRE, Common Weakness Enumeration, [cwe.mitre.org](http://cwe.mitre.org).
- [21] FIRST, Common Vulnerability Standard, Version 3.0, <https://www.first.org/cvss/>.
- [22] E. B. Nightingale and P. Orwick, "Nineteen cybersecurity best practices used to implement the seven properties of highly secured devices in Azure Sphere," July 2020. [Online]. Available: <https://azure.microsoft.com/en-us/resources/best-practices-for-implementing-seven-properties-in-azure-sphere/>. [Accessed 15 October 2020].
- [23] Databeans, Inc., "Q1 2017 Microcontroller Market Tracker," Reno, NV, 2017.