SHCoT: Secure (and Verified) Hybrid Chain of Trust to Protect from Malicious Software in LightWeight Devices

Abderrahmane Sensaoui, Oum-EL-Kheir Aktouf, David Hely
Univ. Grenoble Alpes, Grenoble INP,LCIS*
**Institute of Engineering Univ. Grenoble Alpes*
Valence, France
{name.surname}@lcis.grenoble-inp.fr

***Abstract*—Looking at the speed by which the software and the hardware evolve separately, there is no surprise that the interactions of the two may result in issues and appearance of back-doors to bypass the existing security. Lately, the hardware/software co-design gained lots of interest in both academia and industry, and proposed multiple hybrid solutions to enhance software/hardware interactions, security, and safety while guaranteeing good performance.**

In this paper, we focus on isolation and attestation to enforce the chain of trust in lightweight devices and detect malicious data and software locally and remotely. We present SHCoT, a hardware/software co-design to renew trust in devices. SHCoT is our first attempt to develop a formally verified hybrid solution to enhance existing solutions in the literature. While the work is still in progress, the first results show a partial verification of the security properties of SHCoT and small hardware/software cost.

***Index Terms*—Isolation, Hybrid Security, Attestation, Malware detection**

I. Introduction

Nowadays, the usage of devices embedding micro-controllers is increasing in the Internet of Things, health care and automotive domains. Most of these devices contain sensitive information like cryptographic keys, intellectual property, and private data. To build a resilient device, we need enough layers of protection, to protect the hardware, the software, the internal and external communications between peripherals and devices.

Not only recent attacks [1]–[4] have decreased users trust in these devices, but also, sophisticated protection mechanisms require more resources because they are designed for high computing devices, which can be costly for small devices. Therefore, we have to create a better chain of trust to restore trust in these devices.

Today, to establish trust in devices, early research relied on techniques that can fall in three categories; hardware only [5]–[7], software only [8], [9], and hybrid [10]–[14] based techniques.

The first ones rely on dedicated hardware to perform attestation during boot time (static attestation) or during run time (dynamic attestation). [5], [7] offer good static solutions but they are more suited for high-performance devices and can be very expensive for micro-controllers.

The software-based attestation techniques are mostly time-based. They rely on suppositions like the exact time of certain operations and silent adversary (it means, during an attestation, only the prover is communicating with the verifier) which are difficult to achieve in practice and may be unrealistic for many applications.

The hardware/software-based methods combine software and hardware blocks to offer attestation. This category is more suitable for micro-controllers as the other categories introduce either high cost or high overhead or both.

Even though multiple works with multiple assumptions and security guarantees have been proposed to offer chain of trust, we observed that some critical points were dismissed. The first point is to completely isolate the root of trust software from the other components. The second point is to protect the root of trust software from In/Out (IO) peripherals based attack [15]–[17]. And finally, the third point, which starts to gain interest lately [18], is the computer-aided [19] verification of the chain of trust.

This paper presents the SHCoT (Secure Hybrid Chain of Trust) which is a first attempt to create a strong and computer-aided verified chain of trust to detect malicious software for lightweight devices. SHCoT is composed of two hardware modules and a security monitor. The two hardware modules are a light version of Toubkal [20] which will be used to protect from IO peripherals based attacks, and the execution aware protection (EAP) which will be used to protect the security monitor from return-oriented programmed (ROP) attacks and privilege escalation attacks. The combination of both guarantee confidentiality of the cryptographic keys, the data processed by the security monitor, its code, and on the top of all, it guarantees a secure chain of trust. Finally, the security monitor offers software services such as local and remote attestation. We target lightweight devices that do not require complex memory management. More specifically, a 32-bit RISC-V based core with a physical memory protection (PMP).

The rest of the paper is structured as follows. In section

978-1-7281-5138-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ISSREW.2019.00107

II, we present the backgrounds and assumptions needed to achieve a highly secure chain of trust. Then, in section III we present our solution and how it adds a new layer of protection to micro-controllers. In section IV, we discuss the formal verification and evaluate the cells' area of the SHCoT. Finally, in section V, we conclude this paper.

II. Background and Goals

A. Background

1) Isolation: By isolation we mean mechanisms that provide compartmentalization of software components. Compartments are separated and protected by a hardware component to prevent propagation of flaws from one compartment to the others. In our case, isolation is achieved with the MPU. This includes another layer of memory security by limiting compartments from accessing any memory address.

2) Chain of Trust: The chain of trust is the process of validating the authenticity and integrity of multiple components one by one. The main goal is to ensure that the running software and hardware can be trusted and run safely without breaking the whole system's security.

3) Attestation: In order to guarantee a strong security, the device should support attestation to verify the authenticity and integrity of code or data state. Trusted computing architectures may offer attestation to establish trust in a certain code or data. There are two types of attestation; local and remote attestation. Local attestation is when a piece of code attests another one embedded on the same device. Remote attestation, is when a piece of code attests another one embedded on a different device.

4) Computer-aided Security Properties Verification: Computer-aided verification targets the formal analysis of software and hardware systems. The objective is to provide proof of complex security properties. The challenge here is to provide a proof for hardware/software co-design correctness and a proof of isolation completeness. There are different tools to help draw and check the formal model. We will be using NuSMV [21] with the Linear Temporal Logic (LTL) to draw our model and prove our properties. NuSMV is a symbolic model checker for both hardware and software design, it allows to describe a system as a finite state machine (FSM) and prove its properties and theorems.

5) Physical Memory Protection: The Physical Memory Protection (PMP) is a hardware bloc interfaced with RISC-V based cores and responsible for controlling the core memory accesses. It allows only software components with high privileges, like a kernel, to define memory regions and attribute memory access permissions to each region. The PMP is used to restrict some memory regions to the software running under user mode. While the software running in machine mode has access to practically the whole memory area, except when the configuration is locked, then it affects also the machine mode and it cannot be changed until system reset.

B. Threat Model

We assume that an attacker can compromise the OS and gain privileges. OSes are considered untrusted and may have lots of software vulnerabilities.

We consider external malicious peripherals that can connect to the device physically or remotely and can compromise the device.

We assume that an attacker can take control of any existing peripheral that can read/write the system memory, directly or indirectly, using an interconnect bus like TileLink [22], [23]. The attacker can have access to debug ports, and they can mount passive attacks like DMA probing. We also assume that the attacker does not perform hardware attacks or software attacks that exploit hardware bugs like fault-injection attacks.

We assume that all communications are untrustworthy and that attackers can eavesdrop traffic and inject malicious code or data. Therefore, all incoming data is considered untrusted.

C. SHCoT goals

SHCoT objectives to achieve strong guarantees in micro-controllers fall into:

- **Strong security monitor isolation:** The security monitor has access to all memories, manages cryptographic keys, and configures protection domains. Breaking into the security monitor is fatal to the micro-controller security. SHCoT must guarantee good protection for the security monitor and its data, from both software and IO based attacks. The security monitor is composed of two parts, the immutable root and the monitor which can be updated securely.
- **Attestation:** The security monitor can be used to attest a code or data but also, before communicating with other devices, they can prove their integrity and verify their software states.
- **Computer-aided design verified:** This is a major goal of SHCoT. We want to verify the correctness of our chain of trust implementation and prove the completeness of our attestation properties.
- **Lightweight and flexible chain of trust:** Another goal of SHCoT is to provide a flexible design for lightweight devices.

III. Overview of SHCoT

In this section, we present two hardware modules Toubkal [20] and the EAP. Toubkal is mainly used to protect from IO peripherals based attacks and the EAP is a hybrid isolation to enforce the security monitor protection. And then, we present the security monitor and its software attestation design.

A. Toubkal Hardware Module

Toubkal is responsible of controlling Masters' memory accesses. Toubkal has a Master Look-aside Buffer (MLB) where it stores memory regions configurations. Figure 1 shows the information needed to configure a single region. There is a bit for each Master. When the bit is set, Toubkal grants access to the corresponding Master to that region, and vice versa.

The start address is a multiple of 32 bytes and aligned to its size. The size must be a power of two. Finally, the lock bit permits locking the configuration slot. when it is set, it cannot be changed until the system reset.

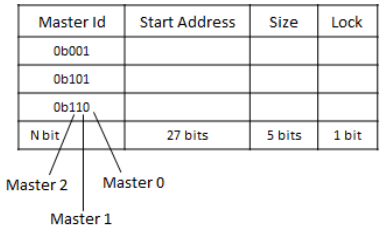

Fig. 1. This figure presents memory regions configuration stored in Toubkal.

Toubkal defines a micro-code area to embed a trusted software that will be able to configure it. In the case of this paper, the micro-code is our security monitor. Toubkal shows some weaknesses concerning the protection of the micro-code [20]. In fact, the micro-code is vulnerable to ROP attacks. However, the EAP is designed in a way the micro-code is protected from ROP attacks as we will see in the next part.

B. The Execution Aware Protection

Execution Aware Protection (EAP) main objective is to guarantee a strong separation of the security monitor from all the rest. It is also combined with the physical memory protected (PMP) to create the D-zones (D for detached zones) which are zones running in user mode and are protected from the privileged mode. EAP offers hybrid isolation, in other words, isolation is implemented by checking the instruction address, the context configuration, and the execution mode, to allow or not a data memory access.

The EAP checks all instructions to control the access to the security monitor and D-zones. It controls three text memory regions; the root, the monitor, and the D-zones. The EAP requires entry points for each one. The main goal of entry points is to force the program to jump into these regions from defined entries. This way, the EAP prevents attackers from randomly calling sensitive functions to extract sensitive data from the protected regions.

The root memory regions, text and data sections, are hard-coded. However, the monitor and D-zones sections can be configured dynamically at run-time. The root is the only trusted software in Toubkal because it is immutable. It is also the one responsible for configuring the EAP to recognize the monitor. The root requires a header at boot time. The header must be located at the top of the Flash and encrypted with a hard-coded symmetric key called K_{SM}. The root decrypts the header using K_{SM} to retrieve the monitor information to verify it and then to configure the EAP.

Configuring D-zones is quite similar. The main difference is just the key used to encrypt and decrypt the header and verify the code is not the same as the one used for the monitor. To verify the integrity and authenticity of D-zones code, the security monitor uses the Key K_{DZ}. This key is not hard-coded as it is in the case of K_{SM}. It is automatically generated from K_{SM}. So whenever we load a library or application into D-zones, the security monitor uses automatically K_{DZ} to attest the code.

For the rest of the code, the security monitor uses another Key K_{RC} to verify applications. And again, this key is generated from K_{DZ}.

TABLE I
MEMORY ACCESS CONTROL FOR ALL LEVELS

From/To	level 0	level 1	level 2	level 3
level 0	rwx	rw	rw	rw
level 1	-	rwx	-	-
level 2	-	-	rwx	rw
level 3	-	-	-	rwx

C. The EAP Memory distribution and access control

The EAP divides the memory into four main groups. Each group has a sensitivity priority. At level zero, which is the most sensitive, there is the memory belonging to the security monitor. At level one, there is the memory belonging to D-zones. At level two, there is memory belonging to the privileged world. And finally, at level three, which is the least sensitive, there is memory belonging to the user world.

Levels zero and two are running in privileged mode, and levels one and three are running in user mode. For convenience and performance matters, code within level zero has access to all other levels. And code within level two has access to level three. Concerning level zero, the reason behind this choice is obvious; it is part of the critical software enforcing system security. Concerning level two, the reason behind it is justified by the fact that level two is running in privileged mode and might embed an OS. An OS that is managing processes running at level three. Table I summarizes the access control list between levels.

Level zero and one are protected from other levels using a program counter based isolation. Level zero is protected from level one using a program counter based isolation, and level two is protected from level three using execution mode and the PMP based isolation. And finally, the EAP uses the PMP to create multiple separated contexts inside levels one and three.

Concerning level zero and level one, which use a program counter based isolation, we define entry points for each level. The entry points are used to force the outside world (level two and level three) to call available services/functions from the defined entry points. Therefore, the entry points are the only gates to level zero and level one. Doing so prevents ROP attacks by reusing and combining gadgets from level zero and level one to leak secrets and change the system's behaviour. However, it does not prevent an attacker from diverting the system behaviour if an attacker exploits a bug inside the code level. Entry points prevent only ROP attacks being mounted from outside the level.

EAP design allows level zero and level one to have multiple entry points. While this point can be convenient and offers flexibility, it is recommended to have as few as possible. Entry points can be considered attack's entries. Therefore, to reduce the attack surface, we need to have few entry points. Entry points are the only codes that can be executed from outside.

D. Security Monitor

The security monitor runs at the highest privilege and has access to all memories. The security monitor is isolated from all other components; software and hardware, by the EAP and Toubkal. It is composed of two parts; *root* and *monitor*.

The root is immutable because it is stored in the ROM. At boot time, it is the only trusted software in the architecture. Its main function is to verify the integrity and authenticity of the monitor. The root provides the HMAC algorithm to compute a cryptographic hash of the monitor using a hard-coded Key K_{SM}. Using the hash, K_{SM} and a key derivation function `genKey()`, the security monitor generate a cryptographic key K_{DZ} that will be used to verify D-zones code. The same process is repeated to generate a third key K_{RC} to verify all other applications.

The monitor is not trusted at system boot. It is stored in non-volatile memory (e.g. Flash memory). A header must be located at the top of the non-volatile memory. The header is encrypted using K_{SM} and contains the monitor configuration: the start address, entry points, size, data start address, data end address, and a magic word for sanity checks. The root decrypts the header and retrieves the configuration to verify the monitor and then configure the EAP so it recognizes it.

The monitor receives control after the root finishes its work. It offers multiple security services such as secure heap allocator, secure update, MMP and PMP configurator. The monitor can be updated securely. We separate the root from the monitor so we can have a small root and verify it easily. As for the monitor, it can be easily updated in case of issues or new features.

The monitor verifies the D-zones code and the rest of applications with the generated keys K_{DZ} and K_{RC}. It configures the different zones then jumps to the system start entry which is, most of the time, the OS entry.

E. Software attestation

SHCoT offers the possibility to attest any piece of code or data (see figure 2). The security monitor defines an API to call the Verifier. The Verifier, as shown in algorithm 1, takes the start address, the end address, and the HMAC. The cryptographic is retrieved automatically with `VerifyZone()`. It is based on the caller zone. Then, the verifier, with the right key, computes the HMAC and then compares it to the given HMAC. Finally, The Verifier sends back the result of the comparison.

IV. Evaluation

A. Security Properties

We define here the security properties required to guarantee a strong chain of trust. The security properties concern both isolation and attestation. We can divide them into two categories, **safe execution** and **confidentiality**.

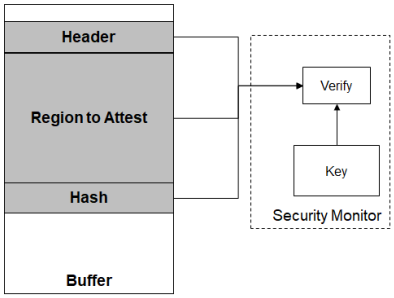

Fig. 2. Software Attestation:

Algorithm 1: Verify function

Input: *start, end*: the start address and the end address of the attested memory region
hmac: the HMAC of the configuration
Output: Success or error code
begin

```
/* Atomic execution is started
   withing the Gateway In          */
/* VerifyZone to retrieve the right
   key                             */
key = VerifyZone();
/* Verify the authenticity of the
   configuration                   */
hmacresult = HACL_HMAC_SHA256(start, end,
 key);
/* compare the computed HMAC and
   the given one                   */
if hmac == hmacresult then
    return success;
else
    return fail;
end
```

end

Safe Execution:

- **SP1 States sequence correctness** The design must respect states' order to be sure it is run safely and securely.
- **SP2 Immutability:** root code has to be immutable, otherwise, an attacker can change it to a malicious code and break the system security. The monitor and secure zones codes have to be semi-immutable. It means the code cannot be changed and run directly. If, we want to update those codes, we have to call the root to check their integrity before calling them.
- **SP3 Controlled Call** Security Monitor and D-zones must be called from the defined entries.
- **SP4 Interrupt Handling** Interrupts can present a high risk to mitigate EAP security. We need to make sure that all interrupts are handled safely.

TABLE II
NOTATIONS USED IN DESIGNING AND VERIFYING THE STATE MACHINE OF EAP

Notation	Description
PC	Program Counter
RC	Root Code
REC	Root Entry Code
MC	Monitor Code
MEC	Monitor Entry Code
DZC	D-zones Code
DZE	D-zones Entry Code
sRomEnt	State when PC is in the Root code entry point
sRomIn	State when PC is legitimately inside the root
sMonitorEnt	State when PC is in the monitor entry points
sMonitorIn	State when PC is legitimately inside the monitor
sDzoneEnt	State when PC is in the D-zones entry points
sDzoneIn	State when PC is legitimately inside the D-zones
sNone	State when PC is in the rest of code
sKill	State when there is a non-authorized access

- **SP5 Configuration** EAP configuration must be possible only from the root, and it is done once and cannot be changed until the system reset.

Confidentiality:

- **SP6 Code confidentiality** Security monitor and D-zones codes are kept confidential from reads attempt from the outside.
- **SP7 Key confidentiality** The hard-coded Key is only accessible from the root. This limits the access scope. The other keys are accessible from the whole security monitor.
- **SP8 Secrets confidentiality** For the security monitor and D-zones, stacks and heaps are reserved respectively and have to be protected from leakage. Access to these regions is only possible from inside each region respectively.

B. The EAP Isolation Verification

We model the EAP as a Finite State Machine (FSM). In this paper, we provide a first attempt to verify and prove the isolation boundaries provided by the EAP. There are eight states in the EAP: sRomEnt, sRomIn, sMonitorEnt, sMonitorIn, sDzoneEnt, sDzoneIn, sNone, sKill. Table II presents the different notations used here. We are using NuSMV [21] to draw our formal model and the Linear Temporal Logic (LTL) specifications to define our isolation and security properties.

In this paper, we give the example of the case where the program is in level two or three, which means the program is in the state sNone. The main goal of the EAP isolation is to completely isolate the security monitor from all the rest. The challenging task is to define all possible transitions correctly while assuring that no secret is leaked. Here, we will focus on proving how the EAP protects from calling gadgets in the security monitor without passing by the secure gateways (entry points).

Figure 3 presents the verified model of the studied case in this paper. The model has four possible states. The sNone state when the PC is running outside levels zero and one. The sMonitorEnt and the sDzoneEnt states when the PC jumps to

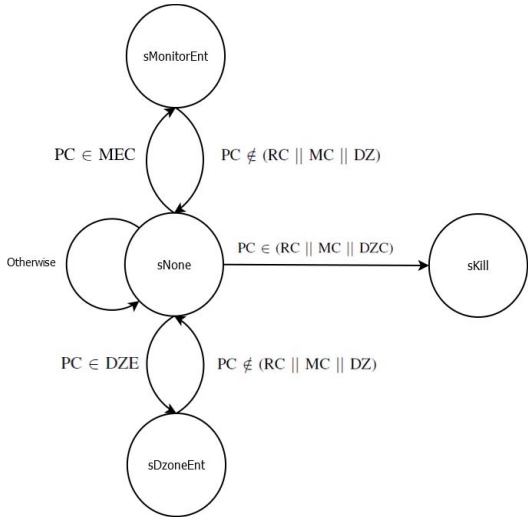

Fig. 3. State sequence correctness starting from the state sNone

an entry of the monitor or the D-zones. And finally, the sKill state, when the PC makes a non-authorized jump to levels zero or one. To guarantee a non-authorized jump to occur, we define the following LTL specification:

```
LTLSPEC G ((states = sNone) & ((PC ∈ MC) & (PC ∈ RC) & (PC ∈ DZC))) → sKill
```

The LTLSPEC G in NuSMV guarantees that the level two and three (state sNone) cannot jump randomly to levels zero and one. They need to call the entry points in order to call a service. This enforces the chain of trust as the only way to establish trust in a software cannot be called randomly, but, it should follow the normal path so an attacker cannot bypass the attestation process.

C. Software Attestation Verification

Our software attestation is built using the HACL* HMAC implementation [24]. HACL* code has been verified and proved correct, memory safe, and secret independent. On top of this, it is a part of the security monitor which is protected by Toubkal and the EAP from ROP and IO based attacks. This guarantees multiple points; controlled calls of the HMAC function, code confidentiality, keys confidentiality and hash calculation intermediate values secrecy. Therefore, **SP3, SP6, SP7** and **SP8** are respected. Besides, the HMAC is precisely part of the root. It means that it is immutable, therefore it guarantees **SP2**.

Finally, concerning state sequence correctness, **SP1**, implies that the HMAC implementation conforms the defined standard specification on all possible inputs, and that it runs in a finite time. These aspects are guaranteed by HACL* implementation [24].

D. Hardware Cost Estimation

We evaluate the hardware area needed to enforce isolation boundaries for a stronger chain of trust. Whether the EAP

TABLE III
HARDWARE STORING COST

Hardware Module	Registers
Toubkal	290
EAP	611

TABLE IV
COMPARISON TO OTHER ARCHITECTURES

	SHCoT	VRASED	SMART	SANCUS
Software/Hardware	Hybrid	Hybrid	Hybrid	Hardware
Isolation	Yes	No	No	Yes
Local Attestation	Yes	No	No	Yes
Remote Attestation	Yes	Yes	Yes	Yes
IO Peripheral Protection	Yes	only DMA	No	No
Software Verification	Partial	Yes	No	No
Hardware Verification	Partial	Yes	No	No

or Toubkal, they have to check every instruction or signal. Therefore, they must respond within the cycle. It has been proven that Toubkal responds within the cycle and does not introduce an overhead. As for the EAP, we use high-speed registers and only combinational logic to guarantee that the checks are done within the cycle. Table III summarizes the cost of needed registers. The EAP requires more registers because it needs to store a symmetric cryptographic key of 256 bit.

E. Comparison with similar architectures

We compare SHCoT to similar architectures. Table IV presents a comparison of four architectures: SHCoT, VRASED [18], SMART [10], and Sancus [14]. First, except for Sancus, the others are all hybrid architectures. Consequently, Sancus results in high cells' area overhead, while the other offers more flexibility, especially SHCoT because the security monitor can be partially updated, while in VRASED and SMART, their root is immutable. SHCoT and Sancus offer isolation, while VRASED and SMART do not. SHCoT, being hybrid, offers more flexibility concerning the number of contexts, while Sancus offer a fixed number of contexts. The four architectures offer attestation. VRASED and SMART offer only remote attestation, while Sancus and SHCoT offer both remote and local attestations. Concerning verification, only VRASED is formally verified. However, the software and hardware were verified separately. SHCoT here is partially verified, and future works aim to complete the whole verification. Finally, concerning IO peripherals attacks, only SHCoT offers this feature. VRASED offers DMA protection but only to the root.

V. CONCLUSION

In this paper, we presented SHCoT, our first attempt to offer a secure and verified chain of trust. Computer-aided verification helps to better design the software and hardware and limit the number of bugs and back-doors. SHCoT offers multiple services like multi-layer isolation, remote and local attestations, and the secure update function. While the first results look promising, there is still work to offer a fully verified hybrid design. The first step for future works is to finish the formal verification of the whole design, and especially the interactions between the hardware and software.

REFERENCES

[1] R. Buhren, S. Gueron, J. Nordholz, J.-P. Seifert, and J. Vetter, "Fault attacks on encrypted general purpose compute platforms," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17. New York, NY, USA: ACM, 2017, pp. 197–204. [Online]. Available: http://doi.acm.org/10.1145/3029806.3029836

[2] M. S. Kate Temkin, "Fusee gelee exploit," https://nvd.nist.gov/vuln/detail/CVE-2018-6242, 2018.

[3] "Shofel2 exploit," https://github.com/fail0verflow/shofel2, 2018.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *USENIX Security Symposium*, 2018.

[5] J. Greene, "Intel ® trusted execution technology hardware-based technology for enhancing server platform security," 2013.

[6] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Löser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, and S. Thom, "ftpm : A firmware-based tpm 2 . 0 implementation," 2015.

[7] A. CORPORATION., "Atmel trusted platform module at97sc3201," 2005.

[8] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla, "Swatt: software-based attestation for embedded devices," *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pp. 272–282, 2004.

[9] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. K. Khosla, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," in *SOSP*, 2005.

[10] K. M. E. Defrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: Secure and minimal architecture for (establishing dynamic) root of trust," in *NDSS*, 2012.

[11] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: a security architecture for tiny embedded devices," in *EuroSys*, 2014.

[12] F. F. Brasser, B. E. Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[13] R. Strackx, F. Piessens, and B. Preneel, "Efficient isolation of trusted subsystems in embedded systems," in *SecureComm*, 2010.

[14] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. V. Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *USENIX Security Symposium*, 2013.

[15] P. Stewin and I. Bystrov, "Understanding dma malware," in *DIMVA*, 2012.

[16] F. L. Sang, V. Nicomette, and Y. Deswarte, "I/o attacks in intel pc-based architectures and countermeasures," *2011 First SysSec Workshop*, pp. 19–26, 2011.

[17] A. T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. W. Moore, and R. N. M. Watson, "Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals," in *NDSS*, 2019.

[18] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Aug. 2019.

[19] Easycrypt, "Easycrypt: Computer-aided cryptographic proofs." https://www.easycrypt.info/trac/, 2009.

[20] A. Sensaoui, D. Hely, and O.-E.-K. Aktouf, "Toubkal: A flexible and efficient hardware isolation module for secure lightweight devices," *(To be appeared) 2019 15th European Dependable Computing Conference (EDCC)*, 2019.

[21] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *CAV*, 2002.

[22] H. C. SiFive, "Diplomatic design patterns : A tilelink case study," 2017.

[23] I. SiFive, "Sifive tilelink specification," 2017.

[24] J. K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "Hacl*: A verified modern cryptographic library," in *ACM Conference on Computer and Communications Security*, 2017.