Opinion: Advancing Remote Attestation via Computer-aided Formal Verification of Designs and Synthesis of Executables

Karim Eldefrawy
SRI International
Manlo Park, California
karim.eldefrawy@sri.com

Gene Tsudik
UC Irvine
Irvine, California
gts@ics.uci.edu

ABSTRACT

Remote Attestation ($\mathcal{R}$A) of embedded/smart/IoT devices is a very important issue on today's security landscape. $\mathcal{R}$A enables a verifier to measures the current internal memory state of an untrusted remote device (prover). $\mathcal{R}$A helps the verifier establish a static or dynamic root of trust in prover. Despite much prior work, state-of-the-art $\mathcal{R}$A techniques unfortunately still lack any solid foundation and offer no ironclad security, safety or robustness guarantees. This paper argues that computer-aided formal verification, and synthesis of executables, of $\mathcal{R}$A protocols and hybrid (software-hardware) architectures is required and currently unaddressed. We believe that this is achievable with current (computer-aided) formal methods frameworks and tools, and that this can help advance and mature $\mathcal{R}$A research if used to establish more rigorous and clear security arguments. To support our opinion, we highlight several examples where subtle issues were missed in the design and security analysis of $\mathcal{R}$A techniques. Despite deceptive simplicity of such protocols, manual analyses and ad hoc implementations often lead to over-simplification of (and subsequent glossing over) important details in the underlying processor and system architectures. Computer-aided formal verification forces a more scrupulous and disciplined consideration of such details, since, otherwise, verification simply fails. The key objective of the research direction we propose is to increase confidence in correctness and security guarantees of current and future $\mathcal{R}$A techniques and their implementations.

KEYWORDS

Secure Remote Attestation, Formal Methods, Formal Verification

ACM Reference Format:
Karim Eldefrawy and Gene Tsudik. 2019. Opinion: Advancing Remote Attestation via Computer-aided Formal Verification of Designs and Synthesis of Executables. In *12th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '19), May 15–17, 2019, Miami, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3317549.3323403

1 INTRODUCTION

In recent years, the number and variety of special-purpose computing devices has grown dramatically, and surpassed general-purpose computers. This includes all kinds of smart and embedded devices, cyber-physical systems (CPS) as well as Internet-of-Things (IoT) gadgets. They can be increasingly found in various settings, such as homes, offices, factories, vehicles and public venues, as well as on, and within, human bodies. They also represent natural and attractive attack targets for malware [? ? ? ?]. Unfortunately, security is typically not the most pressing issue for low-to-medium-end device manufacturers, due to costs, size or power constraints, as well as the usual rush-to-market syndrome. It is thus unrealistic to expect such (especially, low-end) devices to be equipped with means to prevent attacks. The next best thing is detection of compromise or malware presence, which typically requires some form of **Remote Attestation ($\mathcal{R}$A)**. $\mathcal{R}$A is a security service that measures the current internal memory state (i.e., RAM and/or flash) of an untrusted remote device (prover or $\mathcal{P}$rv) by a trusted entity (verifier or $\mathcal{V}$rf). If $\mathcal{V}$rf detects malware presence, $\mathcal{P}$rv's software can be re-set or rolled back and out-of-band measures can be taken to prevent similar infections. In general, $\mathcal{R}$A helps $\mathcal{V}$rf establish a static or dynamic root of trust in $\mathcal{P}$rv and can be used to construct other security services, such as software updates [?], and verified reset and secure erasure [?].

Overall understanding of $\mathcal{R}$A requirements, limitations, and challenges has matured in recent years due to the development of, and attacks on, $\mathcal{R}$A techniques. (See Section 2.2). Despite substantial progress, an important missing component is the high-assurance and rigor derivable from utilizing computer-aided formal verification, and synthesizing correct-by-construction executables, to guarantee security and correctness of $\mathcal{R}$A designs and implementations.

2 BACKGROUND AND PRELIMINARIES

This section overviews $\mathcal{R}$A concepts and prior results, followed by some background on computer-adided verification in the $\mathcal{R}$A context.

2.1 Overview of Remote Attestation

As mentioned earlier, $\mathcal{R}$A allows a trusted verifier ($\mathcal{V}$rf) to remotely measure the memory state of an untrusted remote device ($\mathcal{P}$rv). To start, we focus on the original $\mathcal{R}$A approach of measuring instantaneous integrity of $\mathcal{P}$rv's memory region; other run-time attestation flavors [? ?] are left for future research. [R31B-C3] We also assume that $\mathcal{R}$A requirements as well as correctness and security definitions are complete and formally modeled, perhaps even manually. This can be achieved by modeling the device/architecture being attested, and tying it to a security definition (e.g., via a security game) that captures $\mathcal{R}$A requirements. Such security definitions must precisely capture the requirements. This is indeed the approach used in recent work that attempted to develop a formally verified hybrid $\mathcal{R}$A

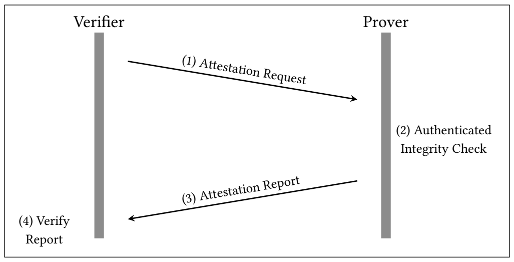

Figure 1: Overview of an abstract $\mathcal{R}$A protocol

design [?]. Our goal is to ensure that a given $\mathcal{R}$A design and its implementation satisfy security requirements. Verifying completeness of initial requirements and security definitions is an important, albeit orthogonal, issue, and is thus out of scope of this paper.

$\mathcal{R}$A is typically realized as a challenge-response protocol, as shown in Figure 1:

(1) $\mathcal{V}$rf sends an attestation request containing a challenge ($\mathcal{C}$hal) to $\mathcal{P}$rv. It may contain a token derived from a secret for $\mathcal{P}$rv to authenticate $\mathcal{V}$rf. (This last step is used only if $\mathcal{V}$rf authentication is desired.)
(2) $\mathcal{P}$rv receives the request, optionally verifies it, and computes an *authenticated integrity check* over the memory to be attested and the $\mathcal{C}$hal. The target memory region might be either pre-defined, or explicitly specified by $\mathcal{V}$rf in step (1) and included in the measurement. In the latter case, authentication of $\mathcal{V}$rf in step (1) is important for overall security of $\mathcal{P}$rv, since the request can specify arbitrary memory regions.
(3) $\mathcal{P}$rv returns the result to $\mathcal{V}$rf.
(4) $\mathcal{V}$rf receives the result and checks whether it corresponds to a valid memory state.

The returned measurement, i.e., challenge-based *authenticated integrity check*, can be realized as a Message Authentication Code (MAC) over $\mathcal{P}$rv's memory. Computing a MAC requires $\mathcal{P}$rv to have a unique secret key ($\mathcal{K}$) shared with $\mathcal{V}$rf.[1] $\mathcal{K}$ must reside in secure storage, inaccessible to all software running on $\mathcal{P}$rv, except for trusted and immutable attestation code – AttCode. Since most $\mathcal{R}$A threat models assume fully compromised software state on $\mathcal{P}$rv, secure storage typically implies some level of hardware support.

There is a range of $\mathcal{R}$A adversarial models. The most common is a remote adversary, i.e., capable of remotely introducing malware onto $\mathcal{P}$rv. Malware is assumed to have complete control of $\mathcal{P}$rv's memory and software. On the other extreme is a physical adversary that can capture, probe, disassemble and/or modify $\mathcal{P}$rv. We refer to [?] for a complete treatment of $\mathcal{R}$A adversarial models. In this paper, we only consider a remote adversary.

2.2 Remote Attestation Landscape

Prior $\mathcal{R}$A designs fall into one of three categories: *software-based, hardware-based, and hybrid. Software-based* (or timing-based) $\mathcal{R}$A is the only viable approach for legacy devices with no hardware security features. Without hardware support, it is (currently) unclear how to guarantee that $\mathcal{K}$ is inaccessible to malware. Therefore, security of software-based approaches [? ?] is loosely attained (or claimed) by setting threshold communication delays between $\mathcal{V}$rf and $\mathcal{P}$rv. Software-based $\mathcal{R}$A is unsuitable for multi-hop and jitter-prone communication, or settings where a compromised $\mathcal{P}$rv is aided (during attestation) by a more powerful accomplice device. It also requires strong constraints and assumptions on the hardware platform and attestation usage [? ?]. *Hardware-based* approaches require $\mathcal{P}$rv's attestation functionality to be housed entirely within dedicated hardware, e.g., SGX [?] or TrustZone [?]. Such hardware features are too expensive for low-end devices, in terms of physical area, energy consumption, and actual cost.

While both hardware- and software-based approaches are not well-suited for settings where low-end devices communicate over the Internet, which is often the case in the IoT world, hybrid $\mathcal{R}$A, based on HW/SW co-design, is a more promising approach. *Hybrid* $\mathcal{R}$A aims to provide the same security guarantees as hardware-based techniques, yet with minimal hardware features. SMART [?] is the first hybrid $\mathcal{R}$A architecture targeting low-end MCUs. In it, AttCode is implemented in software and housed in ROM. SMART 's small hardware footprint guarantees that: (1) AttCode can not be modified, (2) AttCode has exclusive access to $\mathcal{K}$, (3) no part of $\mathcal{K}$ remains in memory after AttCode terminates, and (4) $\mathcal{R}$A runs atomically, i.e., from the first instruction until the last, without interrupts. Property (4) is essential to prevent malware from relocating itself during attestation to evade detection. It also mitigates Return-Oriented Programming (ROP) and similar gadget attacks. A systematic analysis of these properties and their corresponding requirements is attempted in [?], via systematic treatment of $\mathcal{R}$A, starting with a precise definition of the desired service and proceeding to its systematic de-construction into necessary and sufficient properties. These properties are then mapped into a (allegedly minimal) collection of hardware and software components that result in a secure $\mathcal{R}$A architecture.

Despite much progress, a major missing aspect of $\mathcal{R}$A research is high-assurance and rigor obtained by using computer-aided formal methods to guarantee security of a concrete $\mathcal{R}$A design and its implementations. We believe that verifiability and formal security guarantees are particularly important for hybrid $\mathcal{R}$A designs aimed at low- and medium-end devices, due to their rapid proliferation. Massive scale of their deployment translates into equally massive coverage and potentially global impact of attacks. This serves as the main motivation for carefully constructing formally verified $\mathcal{R}$A architectures.

3 COMPUTER-AIDED VERIFICATION & $\mathcal{R}$A

Despite their deceptive simplicity, designing provably secure $\mathcal{R}$A protocols and architectures is challenging. The $\mathcal{R}$A literature includes several protocols and architectures which missed subtle issues that undermine claimed security and correctness guarantees. $\mathcal{R}$A is an ideal first candidate for computer-aided formal verification and synthesis since it only involves two communication rounds and basic cryptographic primitives, e.g., HMAC. As mentioned above, complexity of formal protocol verification quickly becomes prohibitively costly as complexity of underlying cryptographic primitives grows. Given the current state of formal verification tools, $\mathcal{R}$A is within reach, as recently demonstrated in VRASED [?]. VRASED verifies hardware and software components separately. Whereas, ideally both would be verified using the same framework and tools,

[1] Alternatively, $\mathcal{P}$rv could have a unique private key corresponding to a public key held by $\mathcal{V}$rf. Due to higher costs of public key cryptography, and low-end nature of considered devices, we ignore this option, although the discussion below still applies.

in order to obtain a complete computer-aided proof, which increases confidence in its correctness.

3.1 Computer-aided Formal Modeling & Verification

Up to mid 2000-s, computer-aided formal verification and synthesis tools could not handle cryptographic constructs, especially those involving both hardware and software, as is the case with hybrid $\mathcal{R}$A. This changed in the past decade, after verification of several cryptographic primitives and protocols has been demonstrated, e.g., HACL* [?] . Formal modeling and computer-aided verification of security properties is considered [?] to be a key research challenge for the next century. Specifically, security modeling is defined [?] as a generalization of the way cryptography uses precise threat models and security conditions. It is argued that this approach should be used to capture a growing range of security mechanisms encompassing central aspects of cryptography, network security, access control, software system security, hardware security, as well as other branches of the field. While this suggestion may seem obvious, it is not always followed. Lack of rigor and formalization has historically yielded some insecure designs. A prominent example [?] is the process of designing authenticated encryption – a form of encryption that simultaneously ensures both confidentiality and integrity. The issue is the order in which the encryption and authentication operations are performed, such that the combination is secure for any encryption and MAC scheme, as in [?].

There are only a few efforts for formal verification and synthesis of executables in the $\mathcal{R}$A context. The first, HYDRA , utilizes formally verified building block components in a hybrid $\mathcal{R}$A architecture HYDRA [?]. HYDRA builds upon the formally verified `seL4` [?] microkernel to obtain key-protection and memory isolation features that were previously enforced by hardware controls in SMART [?]. However, HYDRA does not formally verify neither hardware modifications nor the software implementation of the attestation executable itself.

The second attempt [?] takes the initial step towards complete formal verification of $\mathcal{R}$A by designing and verifying an architecture called: Verifiable Remote Attestation for Simple Embedded Devices (VRASED). It instantiates a SMART -based hybrid $\mathcal{R}$A co-design and develops several hardware modules verified by modeling their Finite State Machines (FSMs) in Linear Temporal Logic (LTL). Verified LTL modules are automatically synthesized into Verilog to instantiate components realizing required $\mathcal{R}$A properties, which were determined in previous work [?]. (See Section 2.2). VRASED applies these automatically synthesized small hardware modules to the target MCU, and couples them with a ROM that houses $\mathcal{R}$A code implemented using a formally verified binary of a SHA-256-based HMAC from the HACL* library [?].

3.2 Subtle Issues in Prior $\mathcal{R}$A Designs

We discuss here issues in prior $\mathcal{R}$A designs, which undermine security and/or correctness. We argue that computer-aided verification can help discover such problems during the development phase.

(1) **Temporal Consistency:** When an integrity-ensuring function (such as a MAC) is computed over a relatively large input, stability of that input during the entire computation is referred to as *temporal consistency*. This notion was first identified in [?]. Lack of temporal consistency in implementations of MAC or hash functions can lead to non-sensical results and security violations in protocols and systems using them, e.g., $\mathcal{R}$A, verifiable re-sets as well as secure update and erasure. Standard correctness and security definitions of integrity-ensuring functions typically assume that input data (regardless of its size) remains consistent throughout computation. [?] showed that temporal consistency may be lost if another process interrupts execution and modifies portions of input that, either or both: (1) were already processed, or (2) were not processed yet. Such subtleties and discrepancies between (implicit) assumptions in definitions and implementations can be a source of inconsistencies, indeed, temporal inconsistency exists in the TyTan [?] and TrustLite [?] designs as discussed in [?].

(2) **Atomicity of Execution:** $\mathcal{R}$A designs, especially of the hybrid variety, often make assumptions about the underlying main processor architecture and/or instruction set. A common assumption, starting with SMART [?], is that interrupts can be (instantaneously) disabled by AttCode on $\mathcal{P}$rv. This assumption seems reasonable at a first glance. However, in recent work [?], it was shown that, in some micro-controllers, the instruction to enable interrupts consumes several clock cycles and is thus not instantaneous. Hence, this instruction itself can be interrupted. This turns out to be a very subtle issue that inhibits some computer-aided verification proofs from succeeding, and must be handled carefully.

(3) **Availability:** Attacks on availability can be devastating, especially when $\mathcal{R}$A is deployed in settings where $\mathcal{P}$rv serves a real-time or time-critical purpose. For example, SMART [?] includes an API for invoking arbitrary code (specified by $\mathcal{V}$rf) in RAM. This API requires specifying the initial memory location of that code (`return_address`), so that AttCode can jump there after completion. The intent is to use $\mathcal{R}$A as a building block to construct other services, such as secure erasure, reset, and update. It was later discovered that this feature can be abused and cause $\mathcal{P}$rv to be stuck in a loop attesting itself indefinitely. This is because the ROM-resident AttCode does not check whether `return_address` is not the entry point for the AttCode itself. This issue, coupled with other checks (e.g., only executing AttCode from the beginning) to enforce uninterrupted atomic execution, can lead to a major denial-of-service vulnerability. Whereas, formal verification of correctness of the design would have caught this issue, since termination would not have been attainable for this corner case.

4 MOVING FORWARD

Based on the discussion above, we identify the following topics necessary to advance state-of-the-art in $\mathcal{R}$A research.

(1) **Proving Completeness of $\mathcal{R}$A Properties:** Properties required by an $\mathcal{R}$A architecture were previously analyzed and mapped to components [?] (see Section 2.2). An open issue is how to prove completeness and minimality of these properties and components. Proofs of architectural minimality have not been attempted before. Nonetheless, they are highly desirable, especially, in hybrid $\mathcal{R}$A designs, the main

claim-to-fame of which is minimal hardware requirements and modifications to existing hardware, which results in low overall costs and makes them suitable for low-end devices.

(2) **Computer-aided Verification of $\mathcal{R}$A Designs:** While computer-aided verification of cryptographic protocols is still in its early stages, there have been significant developments in recent years. One example is the maturing of EasyCrypt [?], which now allows complex proofs that inter-leave program verification and formalization of mathematical theories (e.g., group theory). EasyCrypt served as the foundation of recent results [?] formally proving security of two protocols based on garbled circuits. We encourage applying EasyCrypt to verify $\mathcal{R}$A protocols, and (ideally) proving composition properties, so as to leverage such work for group settings as proposed in (4) below. In addition, one should consider interactive theorem provers such as PVS [?] and Coq [?] for verifying software/hardware co-designs to obtain an end-to-end mechanically verified $\mathcal{R}$A architectures.

(3) **Correct-by-Construction Implementations of $\mathcal{R}$A:** As mentioned in Section 3, VRASED [?] took the first step towards formal verification of $\mathcal{R}$A by designing and verifying a hybrid $\mathcal{R}$A design for low-end embedded systems. VRASED only synthesized correct-by-construction hardware modules and did not attain the same for the software portion. VRASED relies on a previously verified implementation of SHA-256 based HMAC – HACL*[?] – for the $\mathcal{R}$A executable. Previous work showed [?] that it is possible to synthesize implementations of formally verified cryptographic protocols using EasyCrypt and other toolchains. We suggest the same for $\mathcal{R}$A executables and services using them.

(4) **Heterogenous $\mathcal{P}$rv & Group Settings:** We are unaware of any systematic analysis of properties and requirements for performing $\mathcal{R}$A in group settings with both homogeneous and heterogeneous devices. A first step could be to extend previous single-prover single-verifier setting requirements and properties [?] to groups. Then, similar to (1) and (2) above, computer-aided formal modeling and verification could be used to prove completeness and minimality of these properties. Finally, as outlined in (3), correct-by-construction executables could be extracted from formally-verified protocols for group settings.

ACKNOWLEDGMENTS

We thank Matthias Schunter for shepherding this paper, and Ivan De Oliveira Nunes for suggesting using the term “computer-aided" instead of “automated" in the title and body of this paper.

REFERENCES

[1] The coq proof assistant. `https://coq.inria.fr/`.

[2] Easycrypt: Computer-aided cryptographic proofs. `https://www.easycrypt.info/trac/`.

[3] Pvs specification and verification system. `http://pvs.csl.sri.com/`.

[4] Tigist Abera, N. Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. Invited - things, trouble, trust: on building trust in iot systems. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*, pages 121:1–121:6, 2016.

[5] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. A fast and verified software stack for secure function evaluation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1989–2006, New York, NY, USA, 2017. ACM.

[6] Arm Ltd. Arm TrustZone, 2018.

[7] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. TyTAN: tiny trust anchor for tiny devices. In *DAC*. ACM.

[8] Xavier Carpent, Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Temporal consistency of integrity-ensuring computations and applications to embedded systems security. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 313–327, New York, NY, USA, 2018. ACM.

[9] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. Dynamic integrity measurement and attestation: Towards defense against return-oriented programming attacks. In *Proceedings of the 2009 ACM Workshop on Scalable Trusted Computing*, STC '09, pages 49–54, New York, NY, USA, 2009. ACM.

[10] Karim Eldefrawy, Ivan Oliveira Nunes, Norrathep Rattanavipanon, Michael Steiner, and Gene Tsudik. Formally verified hardware/software co-design for remote attestation. *CoRR*, abs/1811.00175, 2018.

[11] Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. HYDRA: hybrid design for remote attestation (using a formally verified microkernel). In *Wisec*. ACM, 2017.

[12] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In *NDSS*. Internet Society, 2012.

[13] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. A minimalist approach to remote attestation. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[14] SANS Institute. Securing the internet of things survey. https://www.sans.org/reading-room/whitepapers/analyst/securing-internet-things-survey-34785, 2014.

[15] Intel. Intel Software Guard Extensions (Intel SGX).

[16] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[17] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. TrustLite: A security architecture for tiny embedded devices. In *EuroSys*. ACM, 2014.

[18] Xeno Kovah, Corey Kallenberg, Chris Weathers, Amy Herzog, Matthew Albin, and John Butterworth. New results for timing-based attestation. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 2012.

[19] C. Landwehr, D. Boneh, J. C. Mitchell, S. M. Bellovin, S. Landau, and M. E. Lesk. Privacy and cybersecurity: The next 100 years. *Proceedings of the IEEE*, 100(Special Centennial Issue):1659–1673, May 2012.

[20] Ralph Langner. To kill a centrifuge a technical analysis of what Stuxnet's creators tried to achieve, 2013.

[21] Yanlin Li, Yueqiang Cheng, Virgil Gligor, and Adrian Perrig. Establishing software-only root of trust on embedded systems: Facts and fiction. In *Security Protocols—22nd International Workshop*, 2015.

[22] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. Viper: Verifying the integrity of peripherals' firmware. In *CCS*. ACM, 2011.

[23] Wired Magazine. The botnet that broke the internet isn't going away. https://www.wired.com/2016/12/botnet-broke-internet-isnt-going-away/, 2016.

[24] Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. In *ESORICS*, 2010.

[25] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Scuba: Secure code update by attestation in sensor networks. In *ACM workshop on Wireless security*, 2006.

[26] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. *ACM SIGOPS Operating Systems Review*, December 2005.

[27] IEEE Spectrum. The real story of Stuxnet. http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet, 2013.

[28] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A. Sadeghi. Atrium: Runtime attestation resilient under memory attacks. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 384–391, Nov 2017.

[29] Jean-Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. Hacl*: A verified modern cryptographic library. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1789–1806, New York, NY, USA, 2017. ACM.