Remote Attestation via Self-Measurement

XAVIER CARPENT, University of California, San Diego†, USA
NORRATHEP RATTANAVIPANON and GENE TSUDIK, University of California, Irvine, USA

Remote attestation (RA) is a popular means of detecting malware in embedded and IoT devices. RA is usually realized as an interactive protocol, whereby a trusted party (*verifier*) measures software integrity of a potentially compromised remote device (*prover*). Early work focused on purely software-based and fully hardware-based techniques, neither of which is ideal for low-end embedded devices. More recent results yielded hybrid (SW/HW) architectures with a minimal set of features to support efficient and secure RA on low-end devices.

All prior techniques require *on-demand operation*, i.e., RA is performed in *real time*. We identify some drawbacks of this general approach in the context of unattended devices: First, it fails to detect *mobile malware* that enters and leaves prover between successive RA instances. Second, it requires prover to engage in a potentially expensive (in terms of time and energy) computation, which can be harmful for mission-critical or real-time devices.

To address these drawbacks, we introduce the concept of *self-measurement*, whereby prover periodically and securely measures and records its own software state, based on a pre-established schedule. A (possibly untrusted) verifier occasionally collects and verifies these measurements. We present the design of a concrete technique, called Efficient Remote Attestation via Self-Measurement for Unattended Settings, (ERASMUS), justify its features and evaluate its performance. In the process, we also define a new metric, *Quality of Attestation* (QoA). We believe that ERASMUS is well suited for time-sensitive and/or safety-critical applications that are not served well by on-demand RA. Finally, we show that ERASMUS is a promising stepping stone toward handling attestation of multiple devices (i.e., a group or swarm) with high mobility.

CCS Concepts: • **Security and privacy → Embedded systems security**; **Malware and its mitigation**;

Additional Key Words and Phrases: Internet-of-things, malware detection, real-time systems, remote attestation, swarm attestation

ACM Reference format:
Xavier Carpent, Norrathep Rattanavipanon, and Gene Tsudik. 2018. Remote Attestation via Self-Measurement. *ACM Trans. Des. Autom. Electron. Syst.* 24, 1, Article 11 (December 2018), 15 pages.
https://doi.org/10.1145/3279950

11

†Work done while at University of California, Irvine.

This work was supported in part by (1) DHS, under subcontract from HRL Laboratories; (2) ARO under contract W911NF-16-1-0536; and (3) NSF WiFiUS Program Award 1702911.

Authors' addresses: X. Carpent, Jacobs School of Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA; email: xcarpent@gmail.com; N. Rattanavipanon, School of Information and Computer Sciences, University of California, Irvine, 6210 Donald Bren Hall, Irvine CA 92697, USA; email: nrattana@uci.edu; G. Tsudik, School of Information and Computer Sciences, University of California, Irvine, 6210 Donald Bren Hall, Irvine CA 92697, USA; email: gts@ics.uci.edu.

1084-4309/2018/12-ART11 $15.00
https://doi.org/10.1145/3279950

1 INTRODUCTION

In recent years, embedded and cyber-physical systems (CPS), under the guise of *Internet-of-Things (IoT)*, have entered many aspects of daily life, including: homes, office buildings, public venues, factories and vehicles. This trend of adding computerized components to previously analog devices and then inter-connecting them brings many obvious benefits. However, it also greatly expands so-called *"attack surface"* and turns these newly computerized gadgets into natural and attractive attack targets. In particular, as the 2016 Mirai botnet demonstrated, IoT devices can be infected with malware and used as bot-controlled *zombies* in Distributed Denial-of-Service (DDoS) attacks [2]. Also, IoT-borne malware can snoop on device owners (by sensing) or maliciously control critical services (by actuation), as happened with Stuxnet [29].

One key component in securing IoT devices is malware detection, which is typically attained with Remote Attestation (RA). RA is a distinct security service that allows a trusted party, called *verifier*, to securely verify the internal state (including memory and storage) of a remote untrusted and potentially malware-infected device, called *prover*. RA is generally realized as an interactive protocol between prover and verifier. A typical example is described in Reference [6]: (1) verifier sends an attestation request to prover, (2) prover verifies the request[1], (3) computes a cryptographic function of its internal state, (4) sends the result to verifier, and, (5) verifier finally checks the result and decides whether prover is infected.

This general approach is referred to as *on-demand attestation* and all current RA techniques adhere to it. In this article, we identify two important limitations of such approach. First, it is a poor match for unattended devices, since malware that "comes and goes" (i.e., mobile malware [20]) cannot be detected if it leaves prover by the time attestation is performed. Second, for a device working under time constraints (real-time operation) or otherwise providing critical services, on-demand attestation requires performing a possibly time-consuming task while deviating from the device's main function(s).

To address these issues, we design ERASMUS: Efficient Remote Attestation via Self-Measurement for Unattended Settings. ERASMUS is based on *self-measurements*. In it, prover measures and records its state at scheduled times. Measurements are stored in prover's *insecure* memory. Verifier occasionally collects and validates these measurements to establish the *history* of prover's state. In this general approach, verifier imposes only negligible real-time burden on prover. It also offers strictly better quality-of-service than prior attestation techniques, because verifier obtains prover's entire history of measurements, since the last verifier request. In other words, ERASMUS de-couples (1) frequency of prover checking, from (2) frequency of prover measurements, which are equivalent in on-demand attestation. Finally, ERASMUS simplifies RA design (in terms of required features) for prover: Authentication of verifier requests is no longer needed, since computational DoS attacks do not arise. This differs from requirements in Reference [6] that stipulate (potentially expensive) prover authentication of all verifier's RA requests.

We also introduce the new notion of *Quality of Attestation* (QoA) that captures: (1) how prover is attested, (2) how often its state is measured, and (3) how often these measurements are verified. It is the temporal analogue of the concept of quality of swarm attestation (QoSA) introduced in Reference [7] in the context of attesting groups of devices.

NOTE: ERASMUS is not intended as a replacement for on-demand attestation. Clearly, for some devices, and in some settings, real-time on-demand attestation is mandatory, e.g., immediately before or after a software update, or in the context of secure erasure/reset. Also, on-demand

[1]Since attestation is a potentially expensive task, this relatively light-weight verification mitigates computational DoS attacks.

attestation may be more flexible, e.g., if verifier is only interested in measuring a fraction of prover's memory. These two approaches are not mutually exclusive and may be used together to increase QoA, specifically, in terms of freshness of the latest measurement.

The last incentive for the self-measurement RA approach is its suitability for highly mobile groups of devices. RA protocols developed for "swarm attestation," e.g., References [3, 7, 13, 22], are designed to efficiently attest groups of interconnected devices on-demand, with a single interaction between verifier and multiple provers. However, such protocols do not work in highly mobile swarms, since on-demand attestation requires topology to essentially remain static during the entire attestation protocol instance, duration of which is dominated by computation on all swarm devices. Since ERASMUS involves virtually no real-time computation for prover, it is more suitable for high-mobility swarm settings.

After overviewing state-of-the-art in Section 2, we introduce ERASMUS and QoA in Section 3. We then compare ERASMUS with on-demand attestation in Section 4. An implementation and experimental results are discussed in Section 5. Issues arising in time-sensitive applications and partial mitigation measures are discussed in Section 6. Applicability of ERASMUS to swarm attestation is considered in Section 7.

2 REMOTE ATTESTATION (RA)

RA aims to detect malware presence by *verifying* integrity of a remote and untrusted embedded (or IoT) device. As mentioned earlier, it is typically realized as a protocol, whereby trusted verifier interacts with remote prover to obtain an integrity measurement of the latter's state.

RA techniques fall into the three main categories. (1) *Hardware-based attestation* [23, 27] uses dedicated hardware features such as a Trusted Platform Module (TPM) to execute attestation code in a secure environment. Even though such features are currently available in personal computers and smartphones, they are considered a relative "luxury" for very low-end embedded devices. (2) *Software-based attestation* [24, 25] requires no hardware support and performs attestation solely based on precise timing measures. However, it limits prover to being one-hop away from verifier, so that round-trip time is either negligible or fixed. It also relies on strong assumptions about attacker behavior [1] and is typically only used for legacy devices where no other RA techniques are viable. (3) Finally, *hybrid attestation* [5, 10, 15], based on a software/hardware co-design, provides RA while minimizing its impact on underlying hardware features.

SMART [10] is the first hybrid RA design with minimal hardware modifications to existing microcontroller units (MCUs). It has the following key features:

- Attestation code is immutable: it is located in, and executed from, ROM.
- Attestation code is safe: Its execution always terminates and leaks no information other than the attestation result (token).
- Attestation is atomic: (1) It is uninterruptible, and (2) it starts from the first instruction and exits at the last instruction. This is realized in SMART by using hard-wired MCU access controls and disabling interrupts on entering attestation code.
- A secret key (K) is stored in a secure memory location where it can be accessed only from within the attestation code: K is stored in ROM and is guarded by specialized MCU rules.

Reference [6] extended SMART to defend against denial-of-service (DoS) attacks that try to impersonate verifier. This extended variant (referred to as SMART+) additionally requires prover to have a Reliable Read-Only Clock (RROC), needed to perform verifier authentication and prevent replay, reorder, and delay attacks. To ensure reliability, RROC must not be modifiable by software. In SMART+, upon receiving verifier request, ROM-resident attestation code

checks the request's freshness using RROC, authenticates it, and only then proceeds to perform attestation.

TrustLite [15] security architecture also targets RA for low-end devices. It differs from SMART in two ways: (1) Interrupts are allowed and handled securely by the CPU Exception Engine, and (2) access control rules can be programmed using an Execution-Aware Memory Protection Unit (EA-MPU). TyTAN [5] adopts a similar approach while providing additional real-time guarantees and dynamic configuration for safety- and security-critical applications.

HYDRA [11] is a hybrid RA design for medium-end devices devices with a Memory Management Unit (MMU). It builds on a formally verified microkernel, seL4 [14], to ensure memory isolation and enforce access control to memory regions. Using these formally and mathematically proven features, access control rules can be implemented in software and enforced by seL4. Consequently, HYDRA stores K and attestation code in writable memory regions (e.g., flash or RAM) and configures the system such that no other process, besides the attestation process, can access those memory regions. Access control configuration in HYDRA also involves the attestation process having exclusive access to its thread control block as well as to memory regions used for K-related computations. The latter ensures the key protection property. To ensure atomic execution, HYDRA runs the attestation process as the *initial user-space process* with the highest scheduling priority, while the rest of user-land processes are spawned by the attestation process, with lower priorities. Finally, hardware-enforced secure boot is used to provide integrity of seL4 and the attestation process at system initialization time.

In this article, we use SMART+ and HYDRA as the base security architecture for ERASMUS. However, ERASMUS should be equally applicable to other on-demand RA techniques, including aforementioned TrustLite [15] or TyTan [5].

3 SELF-MEASUREMENTS

As discussed in Section 1, all current RA techniques perform *on-demand attestation*. This can be a time-consuming activity that diverts prover's attention away from its primary mission. However, prover performs no RA-related computation between successive verifier's requests.

In contrast, ERASMUS divides RA into two phases. In the *measurement* phase, prover performs self-measurements based on a pre-established schedule, and stores the results. In the collection phase, verifier (whenever it chooses to do so) contacts prover to fetch these measurements. The collection phase is very fast, since it requires practically no computation by prover. In particular, since measurements are based on a MAC computed with a key shared between prover and verifier, no extra protection is needed when prover sends measurements to verifier. Furthermore, unlike on-demand RA, there is no threat of computational DoS on prover, and therefore no need to authenticate verifier's requests.

A prover's measurement M_t computed at time t is defined as:

$$M_t =< t, H(mem_t), \text{MAC}_K(t, H(mem_t)) >,$$

where H is a suitable cryptographic hash (e.g., SHA-256) function and mem_t represents prover's memory at time t. Computation of $H(mem_t)$ and MAC is done in the context of the underlying RA architecture, e.g., SMART+ or HYDRA.

From here onward, $\mathcal{V}$rf and $\mathcal{P}$rv are used to denote verifier and prover, respectively. Although ERASMUS assumes a symmetric key K shared between $\mathcal{V}$rf and $\mathcal{P}$rv, a public key signature scheme could be used instead, with no real impact on security of the scheme except for the higher cost of measurements.

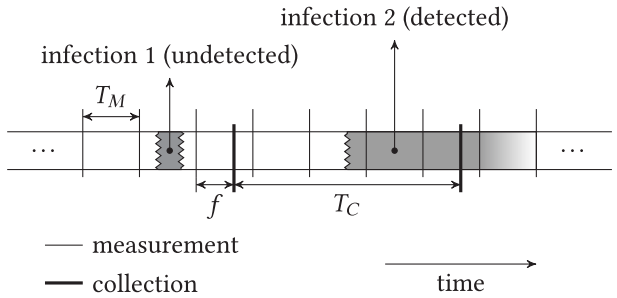

Fig. 1. QoA illustration: Infection 1 by mobile malware is undetected; Infection 2 is detected. T_M is the time between two measurements, T_C is the time between two collections, and f is the freshness of each measurement.

3.1 Quality of Attestation

Quality of Attestation (QoA) is primarily determined by two parameters: (1) time T_M between two successive *measurements* on $\mathcal{P}$rv and (2) time T_C between two successive requests by $\mathcal{V}$rf to collect measurements from $\mathcal{P}$rv.

We assume that, in most cases, $T_C > T_M$. If happens that $T_C \leq T_M$, then $\mathcal{V}$rf simply collects the same measurements more than once, which is redundant. Alternatively, $\mathcal{V}$rf can **explicitly request** $\mathcal{P}$rv to perform a fresh measurement before the collection. In that case, $\mathcal{V}$rf's request would have to be authenticated and checked for freshness (as in SMART+ [6]) before the on-demand measurement is computed. These activities clearly incur additional real-time overhead and delays. This variant is called ERASMUS+OD and it is discussed in Section 4.

Exactly how T_C and T_M are determined depends on the specifics of $\mathcal{P}$rv's mission and its deployment setting. Security impact of these parameters is intuitive. Smaller T_M implies smaller window of opportunity for mobile malware to escape detection. Smaller T_C implies faster malware detection. If either value is large, then attestation becomes ineffective. Meanwhile, though low values increase QoA, they also increase $\mathcal{P}$rv's overall burden, in terms of computation, power consumption and communication.

Without loss of generality, we assume that measurements and collections occur at regular intervals. Of course, in practice this might not work for critical or time-sensitive applications (see Section 6). In fact, it might be advantageous to take measurements at irregular intervals, since doing so might give $\mathcal{P}$rv a bit of an extra edge against mobile malware, as discussed in Section 3.4.

Another ERASMUS parameter is the number of measurements (referred to as k) obtained by $\mathcal{V}$rf in each collection phase. It can range between *one* (only the most recent measurement) and *all*. In a typical setting, $\mathcal{P}$rv's history size should be set such that each measurement is collected exactly once. That is, $k = \lceil T_C/T_M \rceil$.

Finally, the collection phase involves the notion of *freshness*, i.e., how recent is $\mathcal{P}$rv's latest measurement. Depending on the application, maximal freshness might be required, e.g., right before or after a software update. Maximal freshness is attainable via on-demand attestation. In ERASMUS, freshness of a measurement (denoted as f) ranges between T_M and 0, which correspond to minimal and maximal freshness, respectively. On average, we expect $f = T_M/2$.

Figure 1 shows an example with two malware infections. In the first, malware covers its tracks and leaves before any measurement takes place. In the second, malware persists on $\mathcal{P}$rv. Although a measurement occurs perhaps soon after infection, corrective action can be taken only after collection, thus illustrating the importance of a small T_C. Measurements and collections are shown

Table 1. Quality of Attestation Parameters

Notation	Meaning	Tradeoff between
T_M	Time between two consecutive measurements	+ High detection rate − High burden on $\mathcal{P}$rv
T_C	Time between two consecutive $\mathcal{V}$rf requests	+ Fast detection by $\mathcal{V}$rf − High burden on $\mathcal{P}$rv and $\mathcal{V}$rf
f	Freshness of the latest measurement	+ Fresh measurement − High burden on $\mathcal{P}$rv

as punctual events in Figure 1. Although they do take some time to complete (measurements, in particular), they are considered negligible in $\mathcal{P}$rv's overall lifecycle (see Section 5). We summarize QoA parameters and their security implications in Table 1.

3.2 Measurements Storage and Collection

A naïve way for $\mathcal{P}$rv to store measurements is to keep track of them indefinitely. However, this will eventually consume a lot of $\mathcal{P}$rv's storage. To this end, ERASMUS uses *rolling measurements*. A fixed section of $\mathcal{P}$rv's insecure storage is allocated as a windowed (circular) buffer for n measurements. The ith measurement is stored at location $L_{i \bmod n}$. However, it is expected that $\mathcal{V}$rf collects measurements sufficiently often, such that no measurement is over-written. That is, the time between successive collections should be at most $T_C \leq n \cdot T_M$.

The interaction between $\mathcal{P}$rv and $\mathcal{V}$rf is very simple: $\mathcal{V}$rf asks for the k latest measurements, which $\mathcal{P}$rv simply reads from the buffer and transmits. The collection phase does not involve any change of state on $\mathcal{P}$rv and returned measurements are not encrypted. (However, recall that they are authenticated since each measurement is a MAC computed using K). It also does not trigger any significant computation on $\mathcal{P}$rv, i.e., in contrast with on-demand attestation, no cryptographic operations are required in the collection phase.

Self-measurements can be stored in $\mathcal{P}$rv's unprotected storage. This allows malware (that is possibly present on $\mathcal{P}$rv) to tamper with measurements, by modifying, re-ordering and/or deleting them. However, since malware (by design of SMART) cannot access K, it cannot forge measurements. Thus, it is easy to see that any tampering will be detected by $\mathcal{V}$rf at the next collection phase and hence malware presence would be noticed. For same reasons, code that handles request parsing as well as storage and transmission of measurements does not need to be executed in a secure environment, or stored in ROM. Code that performs self-measurement, however, must be protected by the underlying security architecture, as in on-demand RA.

Scheduling in ERASMUS can be implemented in a very simple and stateless manner. Let t be the time specified by RROC at measurement M_t, and let T_M be the time between two successive measurements, as configured in $\mathcal{P}$rv. The windowed buffer slot L_i, used to store M_t, is determined by: $i = \lfloor t/T_M \rfloor \bmod n$.

ERASMUS collection protocol is shown in Figure 2. The underlying RA architecture is not involved in the collection phase. Notation ${}^{*}L_j$ refers to contents of memory location L_j. A sample memory layout is shown in Figure 3.

3.3 Security Considerations

Security of the measurement process itself is based on the underlying security architecture, e.g., SMART+ or HYDRA , which (1) provides measurement code with exclusive access to K, (2) ensures non-malleability and non-interruptibility of the measurement code, and (3) performs memory-cleanup after execution.

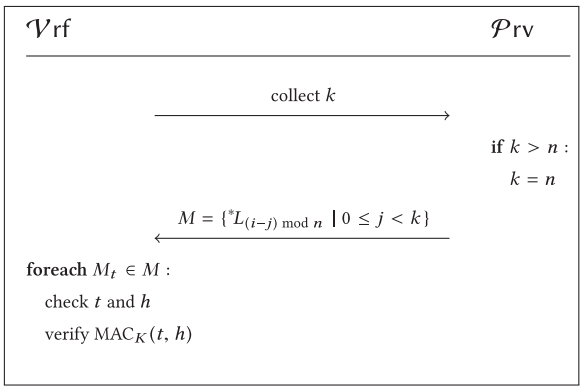

Fig. 2. ERASMUS collection protocol.

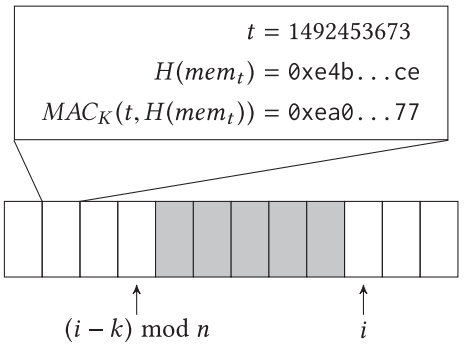

Fig. 3. ERASMUS memory allocation. Example with $n = 12, i = 3, k = 7$.

Timestamps used in the measurement process *must* be obtained from RROC, which (by definition) cannot be modified by non-physical means. This is important, since malware should not influence *when* measurements are taken.

If RROC values could be modified, then the following attack scenario would become possible: malware enters at time t_0 and remains active until a measurement at time $t_0 + \delta$ (with $\delta < T_M$) is taken. Before leaving, malware discards that measurement and resets the counter to t_0. Soon after δ (so that a measurement, valid this time, has been taken for $t_0 + \delta$), malware returns and resets the counter to time elapsed since t_0. Though this example works for one T_M window, it can be extended to arbitrarily many. It requires an additional assumption that no collection took place during the presence of malware.

Fortunately, RROC is already a requirement for SMART+, for a totally different reason: SMART+, RROC helps prevent replay and computational DoS attacks on $\mathcal{P}$rv. Thus, ERASMUS does not require any changes to the underlying security architecture.

As mentioned earlier, measurements need not be stored in protected memory, because tampering is detectable and indicates malware presence on $\mathcal{P}$rv. Likewise, the code to support the collection phase does not require any protection, since measurements are not secret (they are unique for every device and every timestamp value), and their absence or alteration is self-incriminating.

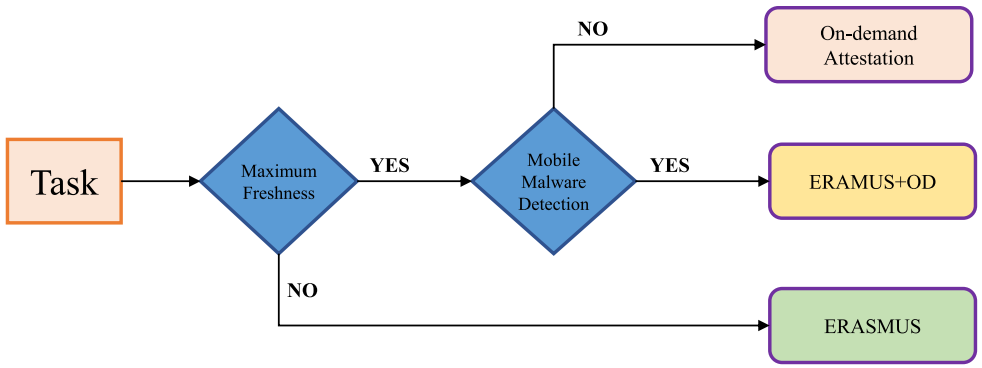

Fig. 4. Diagram summarizing major RA tasks of ERASMUS and on-demand RA.

3.4 Irregular Intervals

A natural extension to ERASMUS is the use of irregular measurement intervals, instead of a fixed T_M. The main motivation is that mobile malware that is aware of fixed scheduling knows when to enter/leave $\mathcal{P}$rv in order to stay undetected. One way to implement irregular intervals is via Cryptographic Pseudo Random Number Generator (CPRNG) [16] initialized (seeded) with the secret key K. Output of the CPRNG can be truncated, such that T_M is upper and/or lower bounded.

For example, after computing M_{t_i}, $\mathcal{P}$rv can set the measurement timer to:

$$T_M^{\text{next}} = \text{map}(\text{CPRNG}_k(t_i)),$$

where map is a function that maps CPRNG output to seconds, e.g., map : $x \mapsto x \bmod (U - L) + L$, with U and L upper and lower bounds, respectively. The timer itself must be read-protected to ensure that T_M^{next} is unknown to malware potentially present on $\mathcal{P}$rv. CPRNG code must be protected the same way as the measurement code.

4 ERASMUS VS. ON-DEMAND RA

We now discuss advantages and disadvantages of ERASMUS as compared to on-demand RA. We also propose a method to combine both techniques while retaining most of their respective benefits. Figure 4 summarizes the discussion throughout this section.

ERASMUS has several advantages over on-demand RA. Most importantly, it enables detection of mobile malware on $\mathcal{P}$rv. Measurements can be performed more often without increased $\mathcal{V}$rf participation. This is an important security consideration, since frequency of (self-)measurements determines the window of opportunity for mobile malware. Another advantage comes from the fact that the collection phase of ERASMUS requires practically no computation on $\mathcal{P}$rv. This makes ERASMUS inherently resilient against computational DoS attacks, without explicitly authenticating $\mathcal{V}$rf's requests. Finally, measurement scheduling in ERASMUS can be made context-aware. This makes ERASMUS better suited for safety-critical applications; this is discussed further in Section 6.

Despite these benefits, ERASMUS does not fully obviate the need for on-demand RA. In applications where a quick reaction to infection is crucial (e.g., immediately before or after a software update or for secure erasure/reset) on-demand RA is still necessary, as it is the only way to maximize freshness of attestation, given that verification is performed immediately after the measurement.

Fortunately, ERASMUS may be combined with on-demand RA to retain advantages of both approaches. This variant, ERASMUS+OD, records $\mathcal{P}$rv's state history to detect mobile malware, and

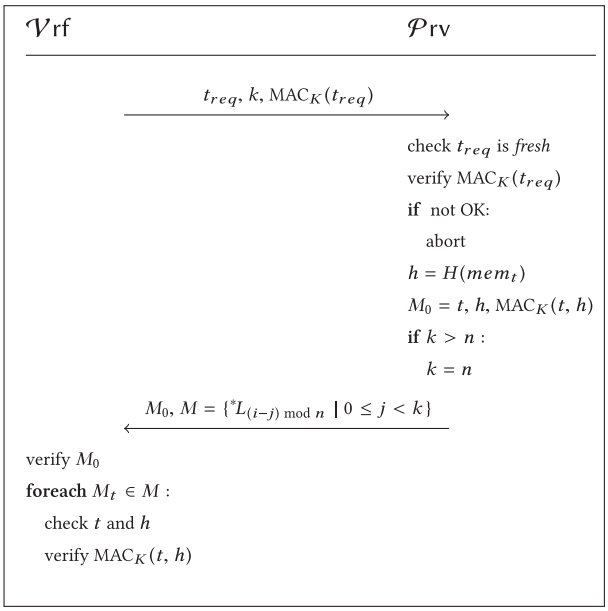

Fig. 5. ERASMUS+OD protocol.

uses on-demand attestation to obtain better freshness. Freshness is particularly relevant whenever real-time attestation is mandatory.

The measurement phase is unmodified, while the collection phase is combined with on-demand attestation request as follows: First, as part of each attestation request, $\mathcal{V}$rf computes and includes an authentication token and specifies k. As in SMART+ [6], authentication of $\mathcal{V}$rf protects $\mathcal{P}$rv against computational DoS. Then, only after checking that a request is valid, $\mathcal{P}$rv computes a new measurement. Finally, this real-time measurement is sent to $\mathcal{V}$rf, along with k previous measurements. This protocol is shown in Figure 5.

5 IMPLEMENTATION

We implemented ERASMUS on two security architectures: SMART+ and HYDRA. The main difference is that the former targets low-end devices, and the latter medium-end devices with a memory management unit (MMU).

5.1 Implementation on SMART+

Figure 6 shows the implementation of ERASMUS atop SMART+ architecture. As in SMART+, measurement code and K reside in ROM. However, the code is invoked periodically and autonomously, whenever a scheduled timer interrupt occurs. We now examine ROM size, hardware costs and runtime.

ROM Size: This greatly depends on the choice of the MAC algorithms. We implement ROM-resident code in "C" using three MAC functions: HMAC-SHA1 [9],[2] HMAC-SHA256 [19], and keyed BLAKE2S [21]. We then use open-source MSP430-gcc compiler [28] to compile the "C" code into an MSP430 executable. Table 2 shows the ROM size for each SMART+-based approach. As expected, ERASMUS requires slightly less ROM than on-demand RA.

[2]Note that HMAC-SHA1 is only used for comparison purposes. We exclude it in our actual implementations due to a recent collision attack on SHA1 [26].

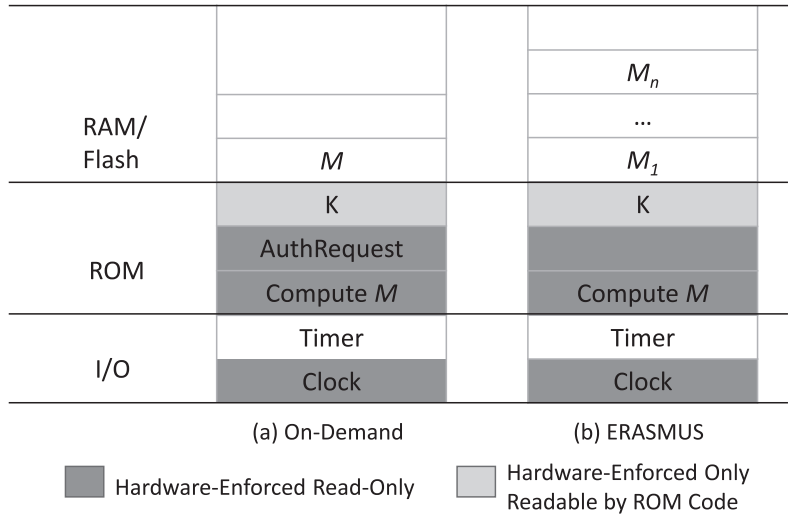

Fig. 6. Memory organization and access rules of SMART+-based RA. r denotes exclusive read-only access.

Table 2. Size of Attestation Executable

MAC Impl.	SMART+		HYDRA	
	On-Demand	ERASMUS	On-Demand	ERASMUS
HMAC-SHA1	4.9KB	4.7KB	—	—
HMAC-SHA256	5.1KB	4.9KB	231.96KB	233.84KB
Keyed BLAKE2S	28.9KB	28.7KB	239.29KB	241.17KB

Table 3. Hardware Cost on MSP430

H/W	Original	On-demand	ERASMUS
Register	579	655	655
Look-up Table	1,731	1,969	1,969

Hardware Cost: We implement the hardware part of ERASMUS by modifying the MSP430 architecture, using the open-source OpenMSP430 core [12]. We modify the memory backbone module in the OpenMSP430 core to support atomic execution of ROM code and exclusive access to K. RROC is realized as a peripheral using a 64-bit register incremented for every clock cycle. To ensure write-protection, a write-enable wire is removed from the RROC module. For timer components, we use the unmodified version of the `omsp_timerA` module provided by OpenMSP430. Note that we do not consider hardware timers as additional hardware cost, because they are common and crucial components of most embedded systems. Indeed, it is unusual to find an embedded device not equipped with at least one timer. Finally, we use Xilinx ISE 14.7 [30] to synthesize our modifications of the OpenMSP430 core from a hardware description language to a combination of registers and look-up tables in an FPGA.

As expected, synthesized results in Table 3 show that ERASMUS utilizes the same number of registers and look-up tables as on-demand RA. Compared to the unmodified OpenMSP430 core, ERASMUS requires roughly 13% and 14% additional registers and look-up tables, respectively.

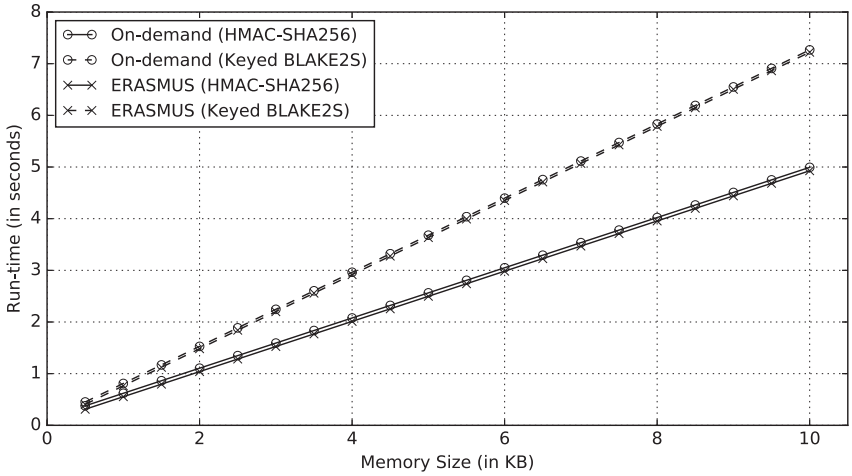

Fig. 7. Measurement Runtime on MSP430-based device @ 8MHz.

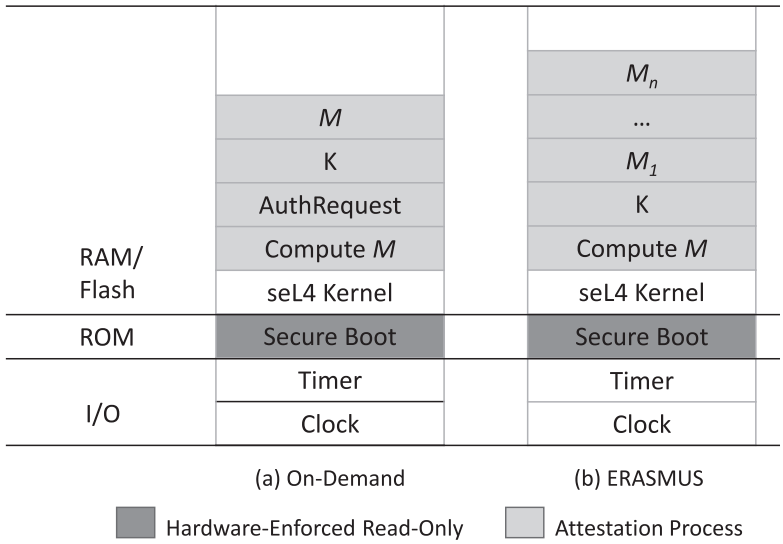

Fig. 8. Memory organization of HYDRA-based on-demand attestation and ERASMUS.

Measurement Runtime: Figure 7 illustrates runtime of the measurement phase for various memory sizes. Not surprisingly, it is linearly dependent on memory size and roughly equivalent to that of on-demand RA.

5.2 Implementation on HYDRA

Figure 8 shows the implementations of HYDRA-based ERASMUS and on-demand RA. Both are realized on an I.MX6 Sabre Lite [4] development board. RROC is implemented based on the software clock approach, suggested by Brasser et al. [6]. Specifically, we use a short-term counter from Sabre Lite's General Purpose Timer (GPT) and our clock code in Pr_{Att} to construct the RROC. When the counter wraps around and causes an interrupt, our clock code handles it by updating higher-order bits of the clock in Pr_{Att}. Then, the clock value is constructed by combining these bits with the GPT

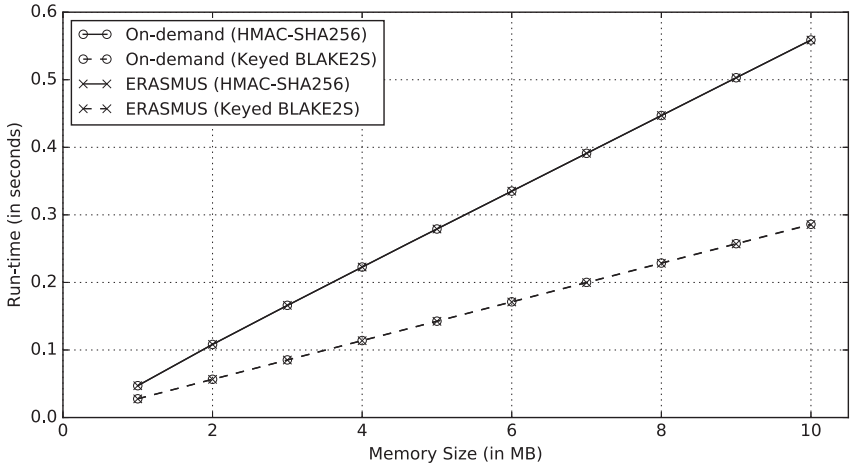

Fig. 9. Measurement Runtime on I.MX6 Sabre Lite @ 1GHz.

Table 4. Runtime (in ms) of Collection Phase on I.MX6-Sabre Lite

Operations	ERASMUS	ERASMUS+OD
Verify Request	N/A	0.005
Compute Measurement[3]	N/A	285.6
Construct UDP Packet	0.003	0.003
Send UDP Packet	0.012	0.012
Total Collection Runtime	0.015	285.6

counter. To ensure the read-only property, Pr_{Att} is given exclusive write-access to RROC components. Also, we use Sabre Lite's Enhanced Periodic Interrupt Timer (EPIT) to schedule execution of ERASMUS measurement code

We base the code of Pr_{Att} on open-source seL4 libraries [17]: seL4utils, seL4vka, seL4vspace, and seL4bench. The first three provide abstractions of process, memory management, and virtual space, respectively, while the last one is used to evaluate performance. Finally, we use Reference [18] to implement the network stack, an Ethernet driver and timer drivers in seL4.[3]

Executable Size: Table 2 compares executable sizes of Pr_{Att} in on-demand RA and ERASMUS. Results show that ERASMUS is only about 1% larger in terms of the executable size. This overhead mostly comes from the need for an additional timer driver.

Measurement Runtime: Measurement runtime of HYDRA-based ERASMUS in Figure 9 follows the same trend as SMART+-based ERASMUS: (1) it is linear as a function of memory sizes, and (2) it is roughly equal to that of on-demand RA. The same figure shows that performing self-measurement (based on keyed BLAKE2S) takes less than 300ms on a 1GHz device with 10MB memory.

Collection Runtime: Table 4 shows the runtime breakdown of the collection phase for each variant. Clearly, runtime of the collection phase in ERASMUS is negligible (by at least a factor of

[3]On 10MB memory using keyed BLAKE2S as the underlying MAC function.

3,000), compared to that of the measurement phase. Collection runtime in ERASMUS+OD, however, is dominated by runtime of performing on-demand attestation.

6 AVAILABILITY IN TIME-SENSITIVE APPLICATIONS

In some cases, it might be undesirable to interrupt execution of $\mathcal{P}$rv's application process to obtain a measurement. This is particularly the case for time-sensitive or safety-critical applications. As discussed in Section 5, measurements can take non-negligible time, e.g., 7s on an 8MHz device with 10KB RAM. Making $\mathcal{P}$rv unavailable for that long is not appropriate.

As is, pure on-demand RA is a poor match for such applications. At the same time, if $\mathcal{P}$rv follows a strict schedule, ERASMUS is also not a remedy, since it suffers from the same issue. However, it can be made more flexible.

One partial measure is for $\mathcal{P}$rv to be self-aware of when time-sensitive tasks occur. That way, it can schedule measurements at appropriate times. If this knowledge is also available to $\mathcal{V}$rf, then on-demand attestation could be used if $\mathcal{V}$rf adapts to $\mathcal{P}$rv's schedule.

We consider another ERASMUS variant: *lenient scheduling*. In it, $\mathcal{P}$rv is allowed to abort the measurement in progress. If something causes a measurement to be aborted, then it can be rescheduled to the end of the current measurement window. However, this comes with some caveats: First, the security architecture needs to be adapted to allow interrupts during measurements. Protection of keys (and cleanup, in case of an interrupt) is still required. Thus, there is still a need for some hardware support. Second, it would be trivial for malware to abort computation of measurements to avoid detection, or simply pretend, when queried by $\mathcal{V}$rf, that all attempted measurements have been aborted. Therefore, $\mathcal{V}$rf must use some external information or policy to decide whether there is a valid justification for each aborted measurement. For instance, $\mathcal{V}$rf can consider that a maximum of w consecutive missing measurements are acceptable. More than that might indicate that a malware purposely aborted measurements to remain undetected. The value of w represents an evident security/availability tradeoff.

These are certainly not ideal measures and the underlying problem seems quite difficult to address deterministically. As is typical for security/usability compromises, real deployment would likely involve policy-based decisions.

7 SWARM ATTESTATION

Some applications require attesting a group (or swarm) of interconnected embedded devices. In such a setting, it is beneficial to take advantage of interconnectivity and perform collective attestation using a dedicated protocol. Several swarm attestation techniques have been proposed. SEDA [3] is the first such scheme, which relies on hybrid attestation security architectures: SMART [10] and TrustLite [15]. SEDA combines them with a request-flooding and response-gathering protocol. SEDA was improved and further specified in LISA [7]. Other related techniques deal with report aggregation [22] or physical attacks [13].

A concept of Quality of Swarm Attestation (QoSA) was introduced in Reference [7] to capture the level of information that $\mathcal{V}$rf obtains as a result of swarm attestation. This can range from binary ("Is the whole swarm healthy?") to full (state of each individual device and topology information). QoA, as introduced in this article, is an orthogonal measure that captures the state of a given device in time. QoA and QoSA can be used in concert with one another.

ERASMUS could be used instead of on-demand attestation in the context of swarm RA protocols. In particular, $\mathcal{P}$rv self-measurements can be coupled with a collection protocol, such as LISA-α, where the latter only relays reports and does not perform any computation. This would yield a clean and conceptually simple approach to swarm attestation, with all the benefits of ERASMUS.

An additional advantage of using ERASMUS in the swarm setting is support for high mobility. Prior swarm RA techniques, such as SEDA, SANA and LISA require swarm topology to remain almost static during the whole swarm attestation instance. This process may be long and prohibitive for applications where connectivity changes often. ERASMUS does not require external input and its collection phase is very fast, since it does not involve any computation; only reading and sending stored measurements. This makes ERASMUS a very natural and viable technique for highly mobile swarms.

Finally, related to the discussion in Section 6, we consider the scenario where availability of at least one in (or a part of) a group of devices is required at all times. This cannot be guaranteed by on-demand swarm attestation, where a large part of the network may be concurrently busy. Meanwhile, with ERASMUS, it is trivial to establish a schedule that ensures that only a fraction of the swarm computes measurements at any given time.

8 CONCLUSION

We designed ERASMUS as an alternative to on-demand RA for low-end devices. ERASMUS provides better QoA, in that it allows $\mathcal{V}$rf to detect mobile malware, which is not possible with on-demand techniques that only detect malware if it is currently on $\mathcal{P}$rv. ERASMUS makes it harder for malware to avoid detection. ERASMUS's other major advantage is that it requires no cryptographic computation by $\mathcal{P}$rv as part of its interaction with $\mathcal{V}$rf. This is particularly relevant in time-sensitive and critical applications, where $\mathcal{P}$rv's availability is very important. We discuss partial mitigation measures for this problem.

We also present a new notion of Quality-of-Attestation (QoA) as a measure of temporal security guarantees given by an attestation technique. We show that timing of measurements and timing of verifications (that are conjoined in on-demand attestation) are two distinct aspects of QoA. They are treated as distinct parameters in ERASMUS. We also discuss the possibility of using on-demand RA as part of ERASMUS collection phase to obtain maximal freshness.

We implemented ERASMUS on two hybrid RA architectures, SMART+ and HYDRA, and demonstrated its viability on both. ERASMUS does not require extra features or a larger ROM than what is needed in SMART+, and each measurement is computed faster than on-demand attestation, since no authentication of $\mathcal{V}$rf requests is needed. Finally, we show that ERASMUS is a promising option for highly mobile groups/swarms of devices, for which no current RA technique works well.

ACKNOWLEDGMENTS

A preliminary (and much shorter) version of this manuscript was published in the proceedings of Design, Automation and Test in Europe (DATE) 2018 [8]. The authors are very grateful to the anonymous reviewers of DATE 2018 and ACM TODAES for their helpful and insightful comments.

REFERENCES

[1] Tigist Abera, N. Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Invited: Things, trouble, trust: On building trust in IoT systems. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'16)*.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security Symposium*.

[3] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable embedded device attestation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*.

[4] Boundary Devices. 2018. i.MX6 ARM Development Board. Retrieved from https://boundarydevices.com/product/sabre-lite-imx6-sbc/.

[5] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. 2015. TyTAN: Tiny trust anchor for tiny devices. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'15)*.
[6] Ferdinand Brasser, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Remote attestation for low-end embedded devices: The prover's perspective. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'16)*.
[7] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. Lightweigh swarm attestation: A tale of two LISA-s. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS'17)*.
[8] Xavier Carpent, Gene Tsudik, and Norrathep Rattanavipanon. 2018. ERASMUS: Efficient remote attestation via self-measurement for unattended settings. In *Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*.
[9] D. Eastlake 3rd and Paul Jones. 2001. *US Secure Hash Algorithm 1 (SHA1)*. Technical Report.
[10] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. 2012. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'12)*.
[11] Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. HYDRA: Hybrid design for remote attestation (using a formally verified microkernel). In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*.
[12] Olivier Girard. 2009. OpenMSP430. Retrieved from https://opencores.org/project/openmsp430.
[13] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. 2016. DARPA: Device attestation resilient to physical attacks. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'16)*.
[14] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. 2009. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*.
[15] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys'14)*.
[16] Yehuda Lindell and Jonathan Katz. 2014. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, London, 68–72.
[17] National ICT Australia and other contributors. 2014. seL4 Libraries. Retrieved from https://github.com/seL4/seL4_libs.
[18] National ICT Australia and other contributors. 2014. util_libs. Retrieved from https://github.com/seL4/util_libs.
[19] National Institute of Standards and Technology. Secure hash signature standard (shs) (fips pub 180-2). 2002.
[20] Rafail Ostrovsky and Moti Yung. 1991. How to withstand mobile virus attacks. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*.
[21] M. J. Saarinen and J. P. Aumasson. 2015. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. Technical Report.
[22] Ahmad-Reza Sadeghi, Matthias Schunter, Ahmad Ibrahim, Mauro Conti, and Gregory Neven. 2016. SANA: Secure and scalable aggregate network attestation. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'16)*.
[23] Dries Schellekens, Brecht Wyseur, and Bart Preneel. 2008. Remote attestation on legacy operating systems with trusted platform modules. *Science of Computer Programming* 74, 1–2 (2008), 13–22.
[24] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. 2004. SWATT: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Research in Security and Privacy (S&P'04)*.
[25] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2006. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of the ACM Workshop on Wireless Security (WiSe'06)*.
[26] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. 2017. The first collision for full SHA-1. In *Advances in Cryptology—CRYPTO 2017*.
[27] Frederic Stumpf, Omid Tafreschi, Patrick Röder, and Claudia Eckert. 2006. A robust integrity reporting protocol for remote attestation. In *Proceedings of the Workshop on Advances in Trusted Computing (WATC'06)*.
[28] Texas Instruments. 2017. MSP430-GCC-OPENSOURCE GCC—Open Source Compiler for MSP Microcontrollers. Retrieved from http://www.ti.com/tool/MSP430-GCC-OPENSOURCE.
[29] Jaikumar Vijayan. 2010. Stuxnet renews power grid security concerns. Retrieved from http://www.computerworld.com/article/2519574/security0/stuxnet-renews-power-grid-security-concerns.html.
[30] Xilinx. 2013. ISE Design Suite. Retrieved from https://www.xilinx.com/content/xilinx/en/downloadNav/design-tools/v2012_4---14_7.html.

Received March 2018; revised August 2018; accepted September 2018