A performance evaluation method for WSNs security

Roberta Daidone
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Pisa, Italy
Email: r.daidone@iet.unipi.it

Gianluca Dini
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Pisa, Italy
Email: g.dini@iet.unipi.it

***Abstract*—The amount of Wireless Sensor Network applications requiring security is getting higher and higher and also developers that are not security experts are often required to secure their applications. Many times they do it without any consciousness of the security performance trade-off arisen by this operation.In this paper we present a method for performance evaluation of a modular security architecture for WSNs. Our method evaluates the costs that have to be paid when introducing security, in terms of memory occupancy, network performance and energy consumption. Knowing these indexes leads to awareness of security costs and helps in fine tuning of security performance trade-offs. A designer may apply our method to know the impact on performance of the security modules he needs. Also, we present performance data collected by applying our method on the implementation of PLASA, a modular security architecture we have designed and evaluated.**

***Index Terms*—security; WSN; performance.**

I. Introduction

Security services typically hinges on cryptographic protocols, which are selected on the basis of security requirements that have to be achieved by the protocol. When the application involves resource constrained devices, a resource greedy security protocol may cause unpredicted delays and/or erratic behaviors and the ability to operate within time bounds could be disrupted. It follows that, even if the control system is originally designed to be stable in presence of network delays, the additional cost due to security may lead to system instability. Knowing security costs in advance is vital in order to trade off security and performance.

In this paper we present a method for performance evaluation of a modular security architecture for WSNs. Our method evaluates the costs that have to be paid when introducing security, independently from the application scenario at hand. The metrics we are considering are memory occupancy, network performance and energy consumption. Knowing these indexes leads to awareness of security costs and helps in fine tuning of security performance trade-offs. Our method is designed for a modular security architecture, so a developer may apply our method to know the impact on performance of the security modules he needs. We describe our method referring to PLASA, a modular and transparent security suite we have designed to provide WSNs with secure communications, key management and secure bootstrapping.

We believe that the method is general enough to be readapted to any other security architecture having modularity characteristics. We apply our method to evaluate the impact of security on performance of PLASA modules and report collected results as a study case.

This paper is structured as follows. In Section III we present the main characteristics of the security architecture we want to evaluate. In Section IV we provide an overview of metrics we have considered for performance evaluation and present the method for evaluating each one of them. In Section V we present performance data collected by applying our method on a real implementation of PLASA. Finally, in Section VI we draw our conclusive remarks.

II. Related work

An important branch of research has focused on the impact of security on performance. For instance, several works have investigated the cost of using off-the-shelf ciphers, encryption modes, and authentication algorithms on wireless sensor nodes in terms of energy consumption, storage and computing overhead. Relevant examples are [1], [4], [6], [7], [3], [5], [9]. Xiao *et al.* and Zhu *et al.* explored first the impact of security on performance [11], [12]. However, these works greatly differ from ours for several reasons. They both investigate the cost of a software implementation of the ciphers, encryption modes, and authentication algorithms. Such an investigation only focuses on the performance implications and does not provide a method for performance evaluation. In contrast, we present a method that is designed to be as general as possible and that can be applied to many platforms and many application scenarios because it is independent from all of them. The set of experiments we performed on real sensor nodes with PLASA are just a validation of our method and should not be considered the only application of the method.

III. PLASA Architecture

PLASA consists of four main components, i) the *Authentication* module, ii) the *KeyDB*, iii) the *Key Management* module, and iv) the *Secure Communication* module. Figure 1 shows these modules stay between the application and the communication stack, but the structure can be easily moved between any other layers of the network stack.

The *Authentication* module is responsible for secure bootstrapping of sensor nodes after deployment. The Authentication module objective is to assure that only authorized nodes can access network communication. Since communications are

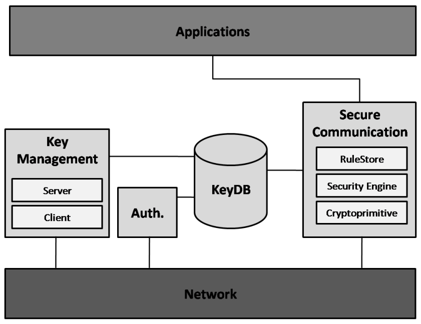

Fig. 1. PLANET security architecture overview.

secured by means of cryptographic keys, the Authentication module aims to distribute cryptographic keys to authorized nodes. To distribute keys in a secure manner, the Authentication module relies on a pre-shared secret that is generally loaded into each network member before network deployment.

The KeyDB is organized as a distributed database to store and retrieve cryptographic keys. It acts as a bridge between different modules of PLASA that access the KeyDB to perform security operations. As shown in Figure 1, its design allows to setup, exchange and refresh cryptographic keys without either creating multiple copies of the same key or causing extra overhead due to key exchange from one module to another. The KeyDB module of each node consists of a *Key Table*. Entries of the Key Table have the same structure: i) the *KeyID field*, used to identify the key within the Table, ii) the *key field*, storing the cryptographic key, and iii) the *Flags field*, used to specify some characteristics of the key usage (e.g. pairwise key, group key, key encryption key etc.).

The *Key Management* module is responsible for refreshing or renewing cryptographic keys periodically or on demand when certain events happen. As shown in Figure 1, this module includes two submodules: i) the server, acting as a Key Manager that broadcasts cryptographic keys to be renewed according to the Key Management protocol, and ii) the client, receiving updates from the Key Manager and taking appropriate actions. The major part of network nodes that implement the Key management module act as clients, while those implementing the server part are special entities, and in general has no other roles.

The *Secure Communication* module is called *STaR* (*Security Transparency and Reconfigurability*). STaR is the heart of PLASA, and extends a work presented in [2]. STaR secures communication in a highly adaptable way, thanks to three layers of data abstraction: i) *traffic flows*, ii) *security policies*, and iii) *labels*. A *traffic flow* is a set of application messages handling the same data or providing the same service. A *security policy* determines what kind of security algorithm is applied by STaR to a certain traffic flow. A *label* determines the mapping between a traffic flow and a security policy. All packets belonging to a given traffic flow can be associated to a common label. Thanks to the label, incoming packets can be unsecured upon being received, according to the security policy associated to the traffic flow they belong to. The STaR component intercepts both incoming and outgoing traffic, segments it into traffic flows, and secures or unsecures them according to the corresponding policies.

STaR provides transparency of security because exports the same interface of the underlying communication stack. Also, STaR assures reconfigurability by allowing users to dynamically change, enable and disable security policies. STaR modular design allows to i) extend it with any other module, ii) move STaR between any layers of the network stack, iii) adapt its interfaces to any communication paradigm.

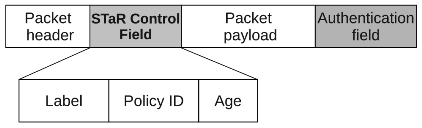

Fig. 2. Example of packet processed by STaR.

Figure 2 shows a packet processed by STaR. The packets arrives to STaR with a header and a payload. Then, STaR builds and inserts the *STaR Control Field* between the header and the payload of the packet. The *STaR Control Field* carries the *Label subfield* which contains the label associated to the traffic flow of the packet, the *PolicyID subfield* which contains the identifier of the security policy associated to the label, and the *Age subfield*, which specifies the age of the mapping of the label to the policy ID. According to the security policy in use, a packet can have its payload encrypted, can be appended with an authentication trailer, or can have both payload and authentication trailer encrypted.

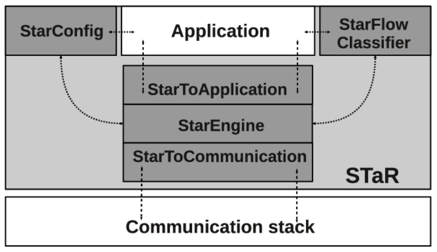

Fig. 3. STaR architecture.

As shown in Figure 3, the STaR component consists in 5 sub-components, namely *StarConfig*, *StarFlowClassifier*, *StarToApplication*, *StarToCommunication*, and *StarEngine*. The *StarFlowClassifier* classifies packets into traffic flows, and determines the associated label. Its implementation consists in a mapping of traffic flows to labels. Application traffic flows can be divided in many ways (e.g. active message types, destination address etc.). The *StarConfig* component allows users to dynamically enable/disable security policies, and change their association to traffic flows, thus providing reconfigurability at runtime. The *StarToApplication* component provides the application with the same communication interface exported by the communication stack. The *StarToCommunication* component makes it possible to connect STaR to the underlying communication stack. The *StarEngine* component encapsulates the mechanism to process security policies.

IV. Performance metrics

In the following we consider four performance metrics that are usually affected by security costs, and provide a method to evaluate them.

The amount of ROM and RAM memory required by security modules may be an issue, especially for devices that cannot load or unload modules at runtime. In these cases, knowing the memory occupancy of each security algorithm and security data structure is vital to assure that the system can accommodate all the needed security material. Also, the amount of entries in security data structures may be a limit for network scalability, so knowing in advance the scalability thresholds allows for designing the network in a way that is more appropriate from the security standpoint.

Security brings in two latency contributions: the *security processing overhead* $\mathrm{d_{proc}}$, and the *security communication overhead* $\mathrm{d_{comm}}$. The security processing overhead consists in a fixed delay due to the processing required by security operations. Generally speaking, such a processing cannot be overlapped with any other operation because it affects the packet structure and involves computationally heavy operations. The security communication overhead consists in the time necessary to transmit the additional fields or packets brought about by security.

The increase in energy consumption due to security is a consequence of the security processing overhead and the security communication overhead. During the extra time required to perform security computations and extra security transmissions, an additional amount of power is consumed. Since WSNs usually involve battery powered devices deployed in unattended environment, such an increase in energy consumption have to be kept into account when evaluating battery characteristics and deployment time.

A. Evaluation method of memory footprint

In order to evaluate memory consumption of PLASA, we take advantage from its modular structure to wire PLASA submodules separately. The wiring method and notation for evaluating memory occupancy is described in the following.

- $\mathrm{M_{Basic}}$ is the image size in bytes of the original network stack, also including the application we want to secure. This value is evaluated experimentally by just removing the wiring of any PLASA module.
- $\mathrm{M_{KeyDB}}$ is the image size in bytes of the KeyDB module. This value is evaluated experimentally by wiring the KeyDB module to the original network stack, obtaining the memory footprint $\mathrm{M_{Output}}$ and then computing $\mathrm{M_{KeyDB}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}}$.
- $\mathrm{M_{Rekeying}}$ is the image size in bytes of the Rekeying module. This value is evaluated experimentally by wiring the KeyDB and Rekeying modules to the original network stack, obtaining the memory footprint $\mathrm{M_{Output}}$ and then computing $\mathrm{M_{Rekeying}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}} - \mathrm{M_{KeyDB}}$.
- $\mathrm{M_{Auth}}$ is the image size in bytes of the Authentication module. This value is evaluated experimentally, starting from the memory footprint $\mathrm{M_{Output}}$ of the original network stack, including the application, and the KeyDB module, wired to the Authentication module. We computed $\mathrm{M_{Auth}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}} - \mathrm{M_{KeyDB}}$.
- $\mathrm{M_{STaR}}$ is the image size in bytes of the STaR module. This value is evaluated experimentally, starting from the memory footprint $\mathrm{M_{Output}}$ of the original network stack, including the application, and the KeyDB module, wired to the STaR module. The STaR module in this case does not include any cryptographic algorithm. We computed $\mathrm{M_{STaR}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}} - \mathrm{M_{KeyDB}}$.
- $\mathrm{M_{Skipjack}}$ is the image size in bytes of the Skipjack submodule that can be provided by the STaR module. This value is evaluated experimentally, starting from the memory footprint $\mathrm{M_{Output}}$ of the original network stack, including the application, and the KeyDB module, wired to the STaR module. The STaR module in this case includes Skipjack. We computed $\mathrm{M_{Skipjack}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}} - \mathrm{M_{KeyDB}} - \mathrm{M_{STaR}}$.
- $\mathrm{M_{SHA-1}}$ is the image size in bytes of the SHA-1 submodule that can be provided by the STaR module. This value is evaluated experimentally, starting from the memory footprint $\mathrm{M_{Output}}$ of the original network stack, including the application, and the KeyDB module, wired to the STaR module. The STaR module in this case includes SHA-1. We computed $\mathrm{M_{SHA-1}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}} - \mathrm{M_{KeyDB}} - \mathrm{M_{STaR}}$.
- $\mathrm{M_{AES}}$ is the image size in bytes of the AES-128 submodule that can be provided by the STaR module. This value is evaluated experimentally, starting from the memory footprint $\mathrm{M_{Output}}$ of the original network stack, including the application, and the KeyDB module, wired to the STaR module. The STaR module in this case includes AES-128. We computed $\mathrm{M_{AES}} = \mathrm{M_{Output}} - \mathrm{M_{Basic}} - \mathrm{M_{KeyDB}} - \mathrm{M_{STaR}}$.

However, the space necessary to allocate executable is not the only storage cost that we have to pay in order to use the security sublayer. The security data structures does not influence ROM occupancy, but RAM occupancy and their size grows with the number of nodes and keys. In general, for RAM occupancy, there is no limit, but the physical capacity, to the amount of memory that a software component may use. However, it is not recommended to fill up the entire RAM with the component variables, because some space is usually needed for the stack. There is no straightforward way to calculate the amount of RAM application needs. However, as a rule of thumb, you should better leave at least 500 byte, or maybe 1 KB, empty. Otherwise you can get a stack overflow and the sensor will hang or do erratic things. Regardless the actual value of the threshold, the important point to capture here is that in memory scarce devices, the amount of memory necessary for security data structures may constitute a limit to the overall system scalability.

B. Evaluation method of processing overhead

The security processing overhead is strictly related on processing capabilities of the device implementing PLASA modules. Our evaluation method considers PLASA modules separately. In order to get an evaluation that is as realistic as possible, we recommend to evaluate such a delay by performing real experiments on real hardware.

The *PLASA processing overhead* time interval is the extra time required to process the packet according to the chosen security policy. Our evaluation method consists in adding a timer module in the original PLASA architecture. The timer instance starts at the beginning of the security operation we want to evaluate, and then stops when the security operation is accomplished. The value collected from the timer can be either printed or inserted as payload of the next packet. Inserting timing information as packet payload is a recommended approach for instance, when the print() function is not supported and data cannot be easily collected. We recommend to consider a confidence interval of at least 95%.

The following are a set of parameter that should be kept into account to have a thorough processing overhead evaluation of the main operations performed by PLASA.

- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{getKey}}$ is the time required by the KeyDB module to retrieve a key from the right entry of the KeyDB.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{setKey}}$ is the time required by the KeyDB module to insert a key as a KeyDB entry.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{nodeInit}}$ is the time needed to initialize the node and generate cryptographic material. Since the needed cryptographic material changes from a protocol another, we recommend to evaluate $\mathrm{d}_{\mathrm{proc}}^{\mathrm{nodeInit}}$ for each protocol provided by the Authentication module.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{initKey}}$ is the time needed to initialize a node with some cryptographic material generated according to the chosen protocol. We recommend to evaluate $\mathrm{d}_{\mathrm{proc}}^{\mathrm{initKey}}$ for each protocol provided by the Key Management module.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{updateKey}}$ is the time needed to refresh some cryptographic material that a node has on board. We recommend to evaluate $\mathrm{d}_{\mathrm{proc}}^{\mathrm{updateKey}}$ for each protocol provided by the Key Management module.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{requestKey}}$ is the time needed to elaborate a key retransmission to recover from key losses.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{secure}}$ is the time needed to secure a packet according to the security operations specified by a certain label. We recommend to evaluate $\mathrm{d}_{\mathrm{proc}}^{\mathrm{secure}}$ for each cryptographic algorithm provided by STaR.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{unsecure}}$ is the time needed to unsecure a packet according to the security operations specified by a certain label. We recommend to evaluate $\mathrm{d}_{\mathrm{proc}}^{\mathrm{unsecure}}$ for each cryptographic algorithm provided by STaR.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{retrieveLabel}}$ is the time needed to retrieve the label associated to the traffic flow of a certain packet.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{retrievePolicy}}$ is the time needed to retrieve the security policy associated to the label in the STaR module.
- $\mathrm{d}_{\mathrm{proc}}^{\mathrm{changePolicy}}$ is the time needed to update the security data structures to change the policy associated to a certain label in the STaR module.

C. *Evaluation method of communication overhead*

The extra transmission overhead can be computed analytically considering the bit rate provided by the hardware we are considering, the amount of extra bits to be transmitted in a secured packet, or the size of extra packets required by security protocols. We evaluate $\mathrm{d}_{\mathrm{comm}}$ as $\mathrm{d}_{\mathrm{comm}} = \text{packet size/bit rate}$

The following are a set of parameter that should be kept into account to have a thorough communication overhead evaluation of the main operations performed by PLASA.

- $\mathrm{d}_{\mathrm{comm}}^{\mathrm{nodeInit}}$ is the time needed to perform transmissions required by the protocol implemented in the Authentication module. Since the initialization protocol is performed *una tantum*, even if we get a high value for this parameter, we consider it as affordable.
- $\mathrm{d}_{\mathrm{comm}}^{\mathrm{updateKey}}$ is the time needed to transmit messages to refresh some cryptographic material that a node has on board. We recommend to evaluate $\mathrm{d}_{\mathrm{comm}}^{\mathrm{updateKey}}$ for each protocol provided by the Key Management module. Also in this case, if we get a high value for this parameter, we consider it as affordable, because this operation is performed sporadically.
- $\mathrm{d}_{\mathrm{comm}}^{\mathrm{requestKey}}$ is the time needed to transmit a key retransmission request to recover from key losses.
- $\mathrm{d}_{\mathrm{comm}}^{\mathrm{secure}}$ is the time needed to transmit the security related fields that are added to secured packet. We recommend to evaluate $\mathrm{d}_{\mathrm{comm}}^{\mathrm{secure}}$ for each cryptographic algorithm provided by STaR.
- $\mathrm{d}_{\mathrm{comm}}^{\mathrm{changePolicy}}$ is the time needed to transmit the reconfiguration packet that triggers the security policy update procedure in the STaR module.

D. *Evaluation method of energy consumption*

The extra energy consumption is a consequence of both security processing and security communication overhead, and can be computed analytically, considering the power consumed by the hardware component that performs the security operations or transmits security-related packets and fields.

We recommend to consider single energy contributions and evaluate each one as $\mathcal{E}_{\mathrm{i}} = \mathcal{P}_{\mathrm{i}} \times \mathrm{d}_{\mathrm{i}}$.

Let d_{i} be the delay due to the considered operation i. Let $\mathcal{P}_{\mathrm{i}} = \mathrm{V}_{\mathrm{i}} \times \mathrm{I}_{\mathrm{i}}$ be the single power contribution, i.e. the product between voltage (V_{i}) and current (I_{i}) of the component that performs the security operation under study. We report a list of the energy consumption contributions of PLASA modules that we recommend to keep into account.

- $\mathcal{E}_{\mathrm{comm}}^{\mathrm{nodeInit}}$ is the energy consumed to perform transmissions required by the protocol implemented by the Authentication module. Since the initialization protocol is performed *una tantum*, a high value for this parameter is considered affordable as it represents the communication overhead of a whole protocol.
- $\mathcal{E}_{\mathrm{comm}}^{\mathrm{updateKey}}$ is the energy consumed to transmit messages to refresh some cryptographic material that a node has on board. We recommend to evaluate $\mathcal{E}_{\mathrm{comm}}^{\mathrm{updateKey}}$ for each protocol provided by the Key Management module.
- $\mathcal{E}_{\mathrm{comm}}^{\mathrm{requestKey}}$ is the energy consumed to transmit a key retransmission request to recover from key losses.
- $\mathcal{E}_{\mathrm{comm}}^{\mathrm{secure}}$ is the energy consumed to transmit the security related fields that are added to secured packet. We recommend to evaluate $\mathcal{E}_{\mathrm{comm}}^{\mathrm{secure}}$ for each cryptographic protocol provided by STaR.
- $\mathcal{E}_{\mathrm{comm}}^{\mathrm{changePolicy}}$ is the energy consumed to transmit the reconfiguration packet that triggers the security policy

update procedure in the STaR module.

- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{getKey}}$ is the energy consumed by the KeyDB module to retrieve a key from the right KeyDB entry.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{setKey}}$ is the energy consumed by the KeyDB module to insert a key as a KeyDB entry.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{nodeInit}}$ is the energy consumed to initialize the node and generate cryptographic material. Since the needed cryptographic material changes from one protocol to another, we recommend to evaluate $\mathcal{E}_{\mathrm{proc}}^{\mathrm{nodeInit}}$ for each protocol provided by the Authentication module.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{initKey}}$ is the energy consumed to initialize a node with the cryptographic material generated according to the chosen protocol. We recommend to evaluate $\mathcal{E}_{\mathrm{proc}}^{\mathrm{initKey}}$ for each protocol provided by the Key Management module.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{updateKey}}$ is the energy consumed to refresh some cryptographic material that a node has on board. We recommend to evaluate $\mathcal{E}_{\mathrm{proc}}^{\mathrm{updateKey}}$ for each protocol provided by the Key Management module.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{requestKey}}$ is the energy consumed to elaborate a key retransmission to recover from key losses.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{secure}}$ is the energy consumed to secure a packet according to the security operations specified by a certain label. We recommend to evaluate $\mathcal{E}_{\mathrm{proc}}^{\mathrm{secure}}$ for each cryptographic algorithm provided by STaR.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{unsecure}}$ is the energy consumed to unsecure a packet according to the security operations specified by a certain label. We recommend to evaluate $\mathcal{E}_{\mathrm{proc}}^{\mathrm{unsecure}}$ for each cryptographic algorithm provided by STaR.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{retrieveLabel}}$ is the energy consumed to retrieve the label associated to the traffic flow which the packet belongs to.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{retrievePolicy}}$ is the energy consumed to retrieve the security policy associated to the label in the STaR module.
- $\mathcal{E}_{\mathrm{proc}}^{\mathrm{changePolicy}}$ is the energy consumed to update the security data structures to change the policy associated to a certain label in the STaR module.

V. Evaluation study case

We have implemented the security features described in Section III for TinyOS 2.1.1, with reference to the Tmote Sky motes [8] and the CC2420 chipset [10]. At the moment, we have implemented the *Skipjack* encryption module and the *SHA-1* module for integrity hashing, AES-128 hardware security.

The amount of ROM memory available on Tmote Sky motes is 48 KB, and may represent a severe constraint while dealing with complex modules like those composing PLASA. In order to evaluate memory consumption on Tmote Sky motes, we considered the TinyOS image size wiring the PLASA submodules separately, according to the method presented in Section IV-A. Table I provides more detailed information on memory occupancy. We believe this amount is reasonable with respect to the available memory. Also, our study cases show that PLASA modular implementation allows for saving memory by loading only a few of the available modules, provided that it is well known what modules are needed.

The *PLASA processing overhead* delay (d_{proc}) has been evaluated experimentally for each of the involved security features of our PLASA TinyOS implementation, according to the method presented in Section IV-B. In our experiments, we observed one sender device at a time transmitting secured packets whose payload is 8 bytes in size. In order to increase the accuracy of our results, we performed 10 repetitions of 20 transmissions for each experiment. Table II provides an overview of $\mathrm{d}_{\mathrm{proc}}$ contributions for PLASA modules. For each row in the table, we report the amount of time required by each PLASA module to perform its operations.

	Memory occupancy (bytes)	**Memory occupancy (%)**
Authentication module	2720	5.67
STaR	1610	3.35
StarEngine (Skipjack)	1748	3.64
StarEngine (SHA-1)	3442	7.17
StarEngine (HW AES-128)	1444	3.01
KeyDB module	198	0.41
Rekeying client	288	0.60

TABLE I
Detailed memory occupancy.

PLASA module	$\mathbf{d_{proc}}$ **(μs)**	**Standard deviation (μs)**
Secure bootstrapping	98804.8	12987.93
STaR	96	0
STaR Skipjack encryption	1217.6	7.16
STaR SHA-1 based authentication	33212.8	30.97
STaR Skipjack encryption and SHA-1 based authentication	34318.4	54.42
STaR hardware AES-128 based encryption and authentication	216	14.22
Rekeying	924.8	22.98

TABLE II
PLASA $\mathrm{d}_{\mathrm{proc}}$ contributions for PLASA modules.

In order to evaluate the transmission overhead analytically, we have considered the time required to transmit different kinds of packets at different stages of the PLASA secured WSN application, with a bit rate of 250 Kb/s and applied the method described in Section IV-C.

Table III provides an overview of the transmission overhead, The value of the secure bootstrapping entry is related to the whole protocol and, since it runs just once after deployment, we consider such contribution as affordable. The d_{tx} contribution of rekeying packets is not negligible, but these packets are transmitted for periodic key refresh only. The rekeying period can be tuned in order to trade-off security and performance.

Energy consumption has been evaluated considering the method reported in Section IV-D and the Tmote sky processing unit and CC2420 components. The values of absorbed current referring to the Tmote sky processing unit and CC2420 components are taken from the respective datasheets [8], [10]. The

PLASA module	$\mathrm{d_{tx}}$ (μs)	Extra bytes
Secure bootstrapping	2976	93
STaR	32	1
STaR Skipjack encryption	256	8
STaR SHA-1 authentication	640	20
STaR Skipjack encryption and SHA-1 authentication	896	28
STaR hardware AES-128 encryption and authentication	512	16
Rekeying	1504	47

TABLE III
PLASA $\mathrm{d_{tx}}$ CONTRIBUTIONS FOR PLASA MODULES.

only exception is the value of the absorbed current during hardware security operations that has been taken from [7].

PLASA module	Processing $\mathcal{P}_{\text{proc}} = 1.08\text{mW}$		Transmission $\mathcal{P}_{\text{tx}} = 31.32\text{mW}$	
	$\mathrm{d_{proc}}$ (μs)	$\mathcal{E}_{\text{proc}}$ (nJ)	$\mathrm{d_{tx}}$ (μs)	$\mathcal{E}_{\text{tx}}$ (nJ)
Secure bootstrapping	98804.8	106709.18	2976	93208.32
STaR	96	103.68	32	1002.24
STaR SkipJack encryption	1217.6	1315.01	256	8017.92
STaR SHA-1 authentication	33212.8	35869.82	640	20044.8
STaR SkipJack encryption and SHA-1 authentication	34318.4	37063.87	896	28062.72
Rekeying	924.8	2.81	1504	47105.28

PLASA module	Processing $\mathcal{P}_{\text{proc}} = 38.14\text{mW}$		Transmission $\mathcal{P}_{\text{tx}} = 31.32\text{mW}$	
	$\mathrm{d_{proc}}$ (μs)	$\mathcal{E}_{\text{proc}}$ (nJ)	$\mathrm{d_{tx}}$ (μs)	$\mathcal{E}_{\text{tx}}$ (nJ)
STaR hardware AES-128 encryption and authentication	216	8238.24	512	16035.84

TABLE IV
PLASA ENERGY CONSUMPTION CONTRIBUTIONS.

Table IV provides an overview of energy contributions for PLASA modules. Considerable contributions in energy consumption for both processing and transmission are due to the standard encryption and authentication algorithms. Note that, while hardware security improves network performance, it has comparable values from the energy consumption standpoint. This is due to the fact that the CC2420 chipset consumes more current than the Tmote Sky processing unit. On the other hand, since hardware security is faster than software security, the reduced time of usage balances the increase of current consumption, allowing us to obtain energy consumptions that are comparable in hardware and software security operations.

VI. CONCLUSION

We have presented a general method to evaluate the impact on performance of security. Our method is deeply described considering PLASA, a modular security architecture we have designed for WSNs. More in details, we have developed a novel approach that mixes experimental and analytical evaluation of each single element of our security architecture. The impact of security on performance is evaluated considering four metrics. Namely, *memory footprint*, *security processing overhead*, *security communication overhead* and *energy consumption*. The method we have presented is as general as possible, and as shown in the study case we have presented, it can be easily applied to any hardware system, architecture or cryptographic module. Of course, when changing one of the aforementioned parameters, we can get values that differ considerably from one case to another. However, the homogeneity of the method guarantees homogeneity of data, so allowing realistic comparisons in terms of performance.

ACKNOWLEDGMENT

This work has been partially supported by PLANET, Platform for the Deployment and Operation of Heterogeneous Networked Cooperating Objects, funded by the European Commission under FP7 with contract number FP7-2009-5-257649 (www.planet-ict.eu) and TENACE, Protecting National Critical Infrastructures From Cyber Threats, funded by the Italian Ministry of Education, University and Research, under the PRIN Framework with contract number 20103P34XC (http://www.dis.uniroma1.it/˜tenace/).

REFERENCES

[1] Chang C.C., Nagel D.J., Muftic S., "Balancing security and energy consumption in wireless sensor networks," in *Mobile Ad-Hoc and Sensor Networks*, ser. Lecture Notes in Computer Science, H. Zhang, S. Olariu, J. Cao, and D. Johnson, Eds. Springer Berlin Heidelberg, 2007, vol. 4864, pp. 469–480.

[2] R. Daidone, G. Dini, and M. Tiloca, "STaR: Security Transparency and Reconfigurability for Wireless Sensor Networks programming," in *Proceedings of the 2nd International Conference on Sensor Networks*, ser. SENSORNETS '13, February 2013.

[3] Ganesan P., Venugopalan R., Peddabachagari P., Dean A. Mueller F., Sichitiu M., "Analyzing and modeling encryption overhead for sensor network nodes," in *Proceedings of the 2nd ACM International conference on Wireless sensor networks and applications WSNA '03*. New York, NY, USA: ACM, 2003, pp. 151–159.

[4] Guimaraes G., Souto E., Sadok D., Kelner J., "Evaluation of security mechanisms in wireless sensor networks," in *Proceedings of the Systems Communications, 2005*, 2005, pp. 428–433.

[5] D. K. Jinwala D., Patel D., "Optimizing the block cipher and modes of operations overhead at the link layer security framework in the wireless sensor networks," in *Proceedings of the 4th International Conference on Information Systems Security ICISS '08*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 258–272.

[6] Law Y.W., Doumen J., Hartel P., "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 1, pp. 65–93, Feb 2006.

[7] K. Lee, J. Kapitanova and S. Son, "The price of security in wireless sensor networks," *Computer Networks*, vol. 54, no. 17, pp. 2967–2978, December 2010.

[8] Moteiv Corporation, "Tmote iv low power wireless sensor module," November 2006. [Online]. Available: http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf

[9] R. Daidone, G. Dini and G. Anastasi, "On Evaluating the Performance Impact of the IEEE 802.15.4 Security Sub-layer," *Computer Communications (to appear)*.

[10] Texas Instruments, "Texas instruments cc2420 2.4 ghz ieee 802.15.4 / zigbee ready rf transceiver," http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2012. [Online]. Available: http://focus.ti.com/lit/ds/symlink/cc2420.pdf

[11] Xiao Y., Chen H.H., Sun B., Wang R., Sethi S., "MAC security and security overhead analysis in the IEEE 802.15.4 wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, pp. 81–81, Apr 2006.

[12] Zhu J., Leina G., Xinfang Z., "Implementation and Time Performance Analysis of Security Suite in LR-WPAN 802.15.4," in *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08*, 2008, pp. 1–5.