A Multi-Layer Moving Target Defense Approach for Protecting Resource-Constrained Distributed Devices*

Valentina Casola[1], Alessandra De Benedictis[1], and Massimiliano Albanese[2]

[1] Department of Electrical Engineering and Information Technology
University of Naples Federico II
Naples, Italy
{casolav,alessandra.debenedictis}@unina.it
[2] Center for Secure Information Systems
George Mason University
Fairfax, VA, USA
malbanes@gmu.edu

Abstract. Techniques aimed at continuously changing a system's attack surface, usually referred to as Moving Target Defense (MTD), are emerging as powerful tools for thwarting cyber attacks. Such mechanisms increase the uncertainty, complexity, and cost for attackers, limit the exposure of vulnerabilities, and ultimately increase overall resiliency. In this paper, we propose an MTD approach for protecting resource-constrained distributed devices through fine-grained *reconfiguration* at different architectural layers. We introduce a coverage-based security metric to quantify the level of security provided by each system configuration: such metric, along with other performance metrics, can be adopted to identify the configuration that best meets the current requirements. In order to show the feasibility of our approach in real-world scenarios, we study its application to Wireless Sensor Networks (WSNs), introducing two different reconfiguration mechanisms. Finally, we show how the proposed mechanisms are effective in reducing the probability of successful attacks.

Keywords: Moving target defense, reconfiguration, proactive security.

1 Introduction

In recent years, we have witnessed a growing interest in techniques aimed at continuously changing a system's attack surface in order to prevent or thwart attacks. This approach to cyber defense is generally referred to as Moving Target Defense (MTD), and it is currently considered one of the *game-changing* themes in cyber security by the Executive Office of the President, National Science and Technology Council [1–3]. As stated in [1], Moving Target Defense *"enables us to create, analyze, evaluate, and deploy mechanisms and strategies that are diverse and that continually shift and change over time to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency"*.

* The work presented in this paper is supported in part by the Army Research Office under award number W911NF-12-1-0448 and MURI award number W911NF-13-1-0421.

The MTD paradigm can be successfully adopted to enforce security requirements in networks composed of distributed and mobile devices, that are typically characterized by limited hardware and software resources. Achieving high levels of security in such constrained environments is not a straightforward task, and innovative approaches must be devised. In this paper we propose an MTD strategy based on fine-grained *reconfiguration* to protect resource-constrained distributed devices, which are characterized by limited processing and storage capabilities, limited battery life, mobility, highly dynamic topology, and frequent failures. Our reconfiguration approach applies to different architectural layers and takes into account not only the hardware and software features of the nodes but also specific security and performance requirements depending on the deployment scenario.

Although changing configuration or system parameters to augment security is a very intuitive principle[3], there is still a lack of metrics to evaluate the security provided by a system and, consequently, quantify the benefits of reconfiguration. To this aim, we introduce a coverage-based security metric to quantify the level of security provided by a given system configuration. Such metric, along with commonly adopted performance and cost metrics, is used to identify the configuration that best meets the current requirements.

In order to show the feasibility of our approach in real applications, we consider Wireless Sensor Networks (WSNs) as a case study. Different mechanisms have been proposed to secure WSNs, but of most such efforts have primarily been aimed at limiting power consumption by reducing the computational and storage requirements. Because of these constraints, the level of security provided by such mechanisms is quite limited, and more complex solutions are not feasible in practice. In this scenario, an MTD approach would make it possible to achieve better security, without requiring computation-intensive solutions, by periodically switching among multiple lightweight cryptosystems. Several reconfiguration mechanisms have been proposed for WSNs [4], mainly based on network reprogramming. They operate at different architectural levels but present similar limitations, as they are battery consuming, introduce a significant overhead, and are potentially not secure.

In order to address these limitations, we introduce two novel mechanisms for reconfiguring sensors that provide better performance from several points of view. We carried out a number of experiments by simulating attack scenarios where an attacker is able to gather partial information on the adopted cryptosystem and attempts a brute force attack. We evaluate the effectiveness of the proposed MTD approach by measuring the probability of successfully completing an attack and show how reconfiguration dramatically decreases such probability.

The paper, which extends the work presented in [5], is organized as follows. Section 2 discusses some of the MTD approaches that have been proposed in the literature, whereas Section 3 introduces our approach and presents the reconfigurable architectural layers we take into account. Subsection 3.1 presents a coverage metric to evaluate the level of security provided by a configuration and in Subsection 3.2 the dependency of security on time is discussed. Section 4 illustrates two innovative reconfiguration

[3] Consider, for instance, the trade-off between the key length in a cryptographic session and the duration of the session itself.

mechanisms for WSNs, whereas Section 5 reports experimental results. Finally, some concluding remarks are given in Section 6.

2 Related Work

The idea behind the Moving Target Defense (MTD) is to change one or more properties of a system in order to present attackers with a varying *attack surface*, so that, by the time the attacker gains enough information about the system for planning an attack, the system's attack surface will be different enough to disrupt it [2, 3]. According to the definition in [6], a system's attack surface is *"the subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack"*. It depends on the system's hardware and software features, and can be changed by dynamically reconfiguring such features at different levels of granularity.

A common MTD practice consists in updating the cryptographic keys used for encryption of communication channels: this introduces some uncertainty for attackers but presents the problem of key distribution, that is a critical phase particularly subject to attacks. More in general, MTD approaches (also referred to as *diversity techniques*) may be applied both at the application level and at a lower level (e.g., code location in memory), as suggested in [7]. Several low-level MTD techniques have been proposed in the literature, based on the idea of automatically generating diverse variants of a program to disrupt vulnerability exploits. A widely deployed example is Address Space Randomization, that was introduced in 2000 by the PAX Team for Linux[4], and has been implemented in most modern operating systems. The basic idea is to randomize the locations of objects in memory so that an attack depending on the knowledge about the address of these objects will fail.

Instruction Set Randomization [8] is another technique for obfuscating the language understood by a system to protect against code-injection attacks: by randomizing the underlying systems instructions, *foreign* code introduced by an attack would fail to execute correctly, regardless of the injection approach.

Another type of low-level diversification is altering how data is stored in memory: in [9] authors present Data Randomization, a technique that provides probabilistic protection against attacks that exploits memory errors by XOR-ing data with random masks. Data randomization uses static analysis to partition instruction operands into equivalence classes: it places two operands in the same class if they may refer to the same object in an execution that does not violate memory safety. Then it assigns a random mask to each class and it generates code instrumented to XOR data read from or written to memory with the mask of the memory operand's class. Therefore, attacks that violate the results of the static analysis have unpredictable results.

Jackson et al. [10] present a diversity technique based on the generation, during the compilation phase, of multiple functionally equivalent machine codes for the same high-level source: with massive-scale software diversity, every user could get its own diversified program version, so that it is impossible for attackers to run a successful attack.

[4] http://pax.grsecurity.net/

The advantage of low-level diversity is that it does not require an understanding of the application's behavior and can be done automatically. However, it is only capable of thwarting specific classes of attacks, such as code injection and memory corruption attacks. Looking at higher-level MTD techniques, several approaches have been proposed, aimed at thwarting the attacker's reconnaissance effort: reconnaissance enables adversaries to gather information about the target system including network topology, configurations, network dynamics. This information can be used to identify system vulnerabilities, and to design and execute specific exploits.

In this regard, several approaches for dynamically changing nodes IP addresses for proactive security have been proposed in the literature, [11–13]. In 2001, Kewley et al. [13] presented a technique called DYNAT (Dynamic Network Address Translation), aimed at confusing any adversary sniffing the network by obfuscating host identity information in TCP/IP packet headers when packets enter public parts of the network. Whenever a client host wants to communicate with a protected server host, the addressing information contained in the header of its request packets is translated (encrypted) by a DYNAT shim before routing the packet to the server. A server gateway receives the packets, reverses the translation in the header fields (decryption) and obtains the true host identity information, used to pass the packets to the target server.

Another work funded by DARPA is presented in [12] by Atighetchi et al., that give an overview of current set of network-level defenses in the DARPA APOD (Application That Participate in Their Own Defense) project. Among the proposed network-centric defense mechanisms, the APOD toolkit also provides a *port and address hopping mechanism*, based on constantly changing a service's TCP identity to both hide the service's real identity and confuse the attacker during reconnaissance. Packets intercepted by attackers will reveal random addresses, which are valid only for a small period of time, e.g., 1 minute. For a port attack to be successful, the attacker must discover the current ports and execute the attack all within one refresh cycle.

Antonatos et al. [14] introduce a proactive defense mechanism called Network Address Space Randomization (NASR) whose objective is to harden networks against worms that use precomputed hitlists of vulnerable targets, by forcing nodes to frequently change their IP addresses. In order to achieve this goal, the authors implemented an advanced NASR-enabled DHCP server to expire DHCP leases at intervals suitable for effective randomization. As the addresses are actually changed at the end-points of a communication, active connections are disrupted during the update; moreover, NASR is limited in the address space as it uses LAN addresses, and requires changes to the end-host operating system, thus making the deployment costly.

In [15] the authors introduce an MTD technique called OpenFlow Random Host Mutation (OF-RHM): each host is assigned an address range, selected from the entire unused address space in the network, and at each mutation interval, a virtual IP is chosen from this range and associated with the host. A Software-Defined Networking (SDN) approach is adopted for range allocation and mutation coordination: a centralized controller (NOX) properly installs flows in OpenFlow switches to forward requests and perform the address translation actions.

Finally, the MTD defense mechanism proposed in [16] is designed to protect the identity of nodes in Mobile Ad Hoc Networks by turning the classical Sybil attack

mechanism into an effective defense mechanism. Legitimate nodes use virtual identities to communicate and periodically change their virtual identity to increase the uncertainty for attackers observing the network. To preserve communication among legitimate nodes, the network layer is modified by introducing a mechanism for mapping virtual identities to real identities, and a protocol for propagating updates of a node's virtual identity to all legitimate nodes.

3 Improving Node Security

In this paper, we propose an MTD framework for reconfiguring resource-constrained devices at different architectural levels, with the reconfiguration granularity chosen at runtime based on current requirements. By reconfiguring a system, it is possible to increase the overall *security level* it provides. Reconfiguration can be either reactive – i.e., the system is reconfigured in response to a detected or perceived threat or new security requirements – or proactive – the system is periodically reconfigured to limit the amount of time each configuration is exposed to malicious observers. Additionally, reconfiguration should be performed in a way to minimize its impact on the system in terms of resource consumption and performance.

Reconfiguration consists in changing one or more of the system's parameters. In our case study focused on WSNs, we identified two main reconfigurable architectural layers:

- *Security layer*. Security in an embedded network can be achieved by implementing a proper cryptosystem. Security layer reconfiguration can be performed by switching among different cryptosystems that satisfy specific security requirements while meeting certain performance and energy consumption constraints.
- *Physical layer*. In embedded systems, the software is part of the node's firmware, that is typically preloaded on internal read-only memory chips (ROM), in contrast to a general-purpose computer that loads its programs into random access memory (RAM) at run-time. Firmware provides the control program of the device and represents the skeleton where different libraries for the implementation of the available cryptosystems and APIs can be plugged and activated via proper software switches. Nodes can be equipped with several versions of the firmware in order to perform physical reconfiguration when needed.

Clearly, further parameters could be considered for reconfiguration, such as the application interface, the hardware configuration or the topology, as long as their reconfiguration is feasible from a technical and energy consumption point of view. In order to perform complex tasks, embedded nodes communicate with one another according to specific application interfaces (APIs), defining the format of the exchanged messages and the communication protocols. Reconfiguration could be applied at this level by providing different APIs for the same application. API reconfiguration could be useful to confuse an attacker that is observing the communication protocol in order to find an exploit to interfere with or control the communication.

As for hardware reconfiguration, it is expensive and not feasible on most of the available devices but needed in case of damage. Network topology could be reconfigured in terms of the view offered to external observers. This could be achieved by

implementing a mechanism that, for instance, presents virtual identities or introduces additional fake nodes into the network. Such a mechanism would need additional protocols and algorithms that are often too expensive for the considered nodes.

The choice of the reconfiguration level impacts both the system's performance and the provided level of security. From the performance point of view, changing the firmware of all the nodes in the network or a subset of them is much more expensive – in terms of latency and power consumption – than changing the cryptosystem, whose reconfiguration could be handled in software. On the other side, by changing the entire application running on a node, it becomes harder for an attacker to exploit software vulnerabilities and gain complete control of the node.

At the security layer, the cryptosystem itself is designed to cope with a specific set of attacks and provides an intrinsic level of security, depending on the cryptographic scheme, the algorithm, the length of the keys, etc. Reconfiguration of the cryptosystem can increase the level of security in two ways, that is by switching to a cryptosystem that covers a larger set of attacks (e.g., to cope with some detected or perceived threats), or by selecting an equivalent cryptosystem that uses different parameters. Given a certain fixed configuration, the more an attacker is able to observe, the more he will be able to infer information about the system. By continuously changing the system's configuration, the attacker will be presented with different views of the system over time, and will have to restart the reconnaissance effort multiple times in order to identify a viable exploit.

Once the admissible configurations have been identified, the selection of the new configuration is performed by a security-driven scheduler. The scheduler can be either a centralized entity making decisions on the global network configuration, or a decentralized component, independently deployed on each network node, making local reconfiguration decisions. In a *centralized* approach, a central entity triggers a configuration update based on some events (e.g., timer expiration, detected security threat) and transmits its decision to all the nodes that are involved. In a *decentralized* approach, each node is able to schedule, independently from other nodes, when to update its own configuration. Communication among legitimate nodes is preserved adopting additional mechanisms, described in details in the following section.

In the remainder of this paper, we will discuss the methodology adopted to evaluate the level of security provided by a system configuration, and how to increase it using a reconfiguration approach.

3.1 Security Level Evaluation

The security of complex systems depends on many technical and organizational issues that must be properly addressed. The need for a clear definition and selection of security rules has led system administrators to set up security policies trying to adopt formal approaches to describe system security configurations. In spite of the ambiguity of such policies, a common approach to evaluate a system's security is through evaluation of its security policy. At present, such an evaluation is performed *by hand* whenever enterprises endeavor to extend their trusted domain and cooperate [17]. This approach also includes well known standards as Common Criteria and TCSEC [18, 19], that are

very suitable to assess and audit the security level provided by a company, by a specific procedure or, in general, by a system.

The Common Criteria (CC) for Information Technology Security Evaluation (Common Criteria or CC) [18] are an internationally approved set of standard for computer security certification. They are used by Government customers in the USA and the NATO community along with other organizations, particularly in the public sector, to determine the level of security and assurance of various technology products. However, the assurance levels provided by CC (from EAL1 to EAL7) do not measure the security of the system itself, but simply state at what level the system was tested, and do not find a direct application in our approach.

Defining a quantitative measure of the level of security provided by a system is a complex task. Several security metrics have been proposed in the literature, mostly based on the analysis of attack graphs or on risk quantification [20–23]. Other approaches, such as the one adopted by the Common Vulnerability Scoring System (CVSS) [24], try to rate the severity of security vulnerabilities and assign a score to a system based on the vulnerabilities it is subject to.

Other metrics are linked to the adopted configurations [25, 26], they are centered on the mechanisms that are available to enforce a subset of security requirements. We started from these considerations to introduce a metric based on the *coverage* of a configuration respect of a set of known attacks. An attack could have several objectives, such as physically taking possession of a node, interfering with communication at the physical level, exploiting software vulnerabilities to take control of a node, disturb network operation at routing/application level or intercept sensitive data. In this discussion we are interested in attacks aimed at interfering, steering or eavesdropping communications at the application layer among nodes, and at exploiting vulnerabilities of the firmware installed on nodes.

Let $Threats$ define the set of threats of interest, belonging to the above discussed set of attacks. A configuration c is said to *cover* a threat $t \in Threats$, if either the cryptosystem implemented at the security layer or the specific firmware version running on the node include mechanisms to protect the node from such threat.

Once the admissible configurations and the attacks of interest have been identified, it is possible to build an *Attack Coverage Table*, that helps define the levels of security provided by each configuration [27]. Table 1 shows an example of attack coverage table relative to configurations $\{c_1, c_2, c_3, c_4\}$, under the hypothesis that three attacks of interest have been identified, namely $AttackA$, $AttackB$ and $AttackC$. An increasing level of security (from L_1 to L_4 in the example) can be assigned to configurations, based on the risk associated with the attacks and their coverage properties.

Attack coverage can be defined either as an ON/OFF property (that is an attack is covered or uncovered), or in terms of the degree of satisfaction of specific requirements (e.g. authentication, integrity, confidentiality, key distribution...), using a scoring system (similar to CVSS for vulnerabilities). Coverage with respect to a specific attack could even be defined in terms of the effort an attacker needs to make the attack succeed.

The level of security associated with a configuration could simply depend on the number of covered threats, or it could be set depending on the risk associated with each threat, either in a static way (the risk associated with a threat is set at deployment and

Table 1. An example of Attacks Coverage Table

Conf	*Attack A*	*Attack B*	*Attack C*	*SL*
c_1	×			L_1
c_2	×		×	L_2
c_3	×	×	×	L_3
c_4	×		×	L_2

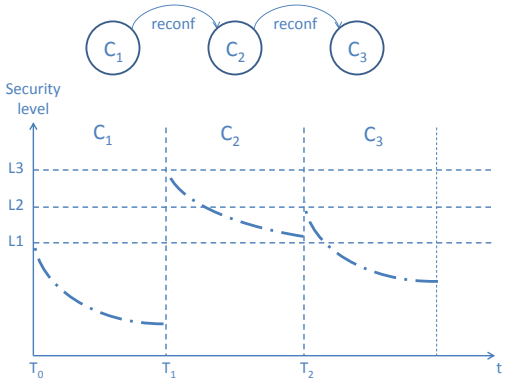


Fig. 1. Reconfigurations and security level

remains unchanged for the entire operation of the network) or dynamically (the risk associated with a threat changes dynamically during network operation depending on current conditions and possible detection events).

3.2 Modeling the Security Level

As previously said, each configuration provides a certain level of security, which depends on the implemented cryptosystem (cryptographic scheme, algorithms, and keys) and is characterized by an intrinsic value. Indeed, the longer a system configuration is exposed to malicious observers, the more the actual level of security decreases. For this reason, the security level is a monotonically decreasing function, with its maximum corresponding to the intrinsic security level associated with the specific implemented cryptosystem.

As illustrated in Fig. 1, using reconfiguration, we can prevent the security level from falling below a certain threshold, and periodically reset it to the intrinsic value associated with a new configuration. Dually speaking, we avoid that the probability of successfully completing an attack increases beyond a certain threshold. In fact, such probability depends on the considered type of attack and is usually represented by a monotonically increasing function: the longer an attacker can try to exploit a system, the higher the success probability is. It is easy to demonstrate that, by introducing re-

configuration, we can *break* the monotonicity, such that the probability of successfully completing an attack actually decreases every time the system is reconfigured.

Theorem 1. *Let $[0, T]$ be an observation interval, and let $n \in \mathbb{N}$ be an integer greater than or equal to 2, representing the number of reconfigurations in $[0, T]$. Then the following inequality holds.*

$$Pr\left(success\left([0, T], n\right)\right) \leq Pr\left(success\left([0, T], 0\right)\right) \tag{1}$$

where $Pr\left(success\left(I, x\right)\right)$ denotes the probability that the attacker is successful within the temporal interval I if x reconfigurations are performed during the same interval. ■

In order to prove Theorem 1, we need to consider that the probability that the attacker will successfully break the cryptosystem between 0 and T when the interval $[0, T]$ is broken down into n validity intervals – and a different cryptosystem is used in each such intervals – can be written as

$$\Pr(success([0, T], n)) = 1 - \Pr(\neg success([0, T], n)) \tag{2}$$

The probability $\Pr(\neg success([0, T], n))$ that the attacker does not succeed by time T is the probability that he does not succeed in any of the n validity intervals.

$$\begin{aligned} \Pr(\neg success([0, T], n)) = \Pr(&\neg success\left(\left[0, \tfrac{1}{n} \cdot T\right]\right) \\ &\wedge \neg success\left(\left[\tfrac{1}{n} \cdot T, \tfrac{2}{n} \cdot T\right]\right) \\ &\wedge \ldots \wedge \neg success\left(\left[\tfrac{n-1}{n} \cdot T, T\right]\right)) \end{aligned} \tag{3}$$

The events $\neg success\left(\left[0, \frac{1}{n} \cdot T\right]\right)$, …, $\neg success\left(\left[\frac{n-1}{n} \cdot T, T\right]\right)$ are clearly independent, thus $\Pr(\neg success([0, T], n))$ can be computed as follows.

$$\Pr(\neg success([0, T], n)) = \prod_{i=0}^{n-1} \left(1 - \Pr\left(success\left(\left[\frac{i}{n} \cdot T, \frac{i+1}{n} \cdot T\right]\right)\right)\right) \tag{4}$$

As the probability that the attacker can break the system in a given interval is directly proportional to the length of the interval itself, we can conclude that, for all $i \in [0, n-1]$, $\Pr\left(success\left(\left[\frac{i}{n} \cdot T, \frac{i+1}{n} \cdot T\right]\right)\right) = \frac{\Pr(success([0,T]))}{n}$. This conclusion relies on the simplifying assumption that the different cryptosystems used in our framework are equivalent in terms of attack time. Generalizing this result to the case of heterogeneous cryptosystems is straightforward, but it is omitted for reasons of space. Additionally, the above conclusion assumes that the interval $[0, T]$ is larger than the time needed to complete a full brute force attack[5]. Then, Equation 4 can be rewritten as follows.

[5] If $\Pr(\neg success([0, T])) = 1$, then there may exist a sub-interval $[t_i, t_j]$ of $[0, T]$ such that $\Pr(\neg success([t_i, t_j])) = 1$.

$$\begin{aligned}\Pr(\neg success([0,T],n)) &= \prod_{i=0}^{n-1}\left(1-\frac{\Pr(success([0,T]))}{n}\right)\\ &= \left(1-\frac{\Pr(success([0,T]))}{n}\right)^n\end{aligned} \tag{5}$$

In order to complete the proof, we need the results of another theorem:

Theorem 2. *Let $x \in [0,1]$ be a real number and let $n \in \mathbb{N}$ be an integer number. The following inequality holds.*

$$\left(1-\frac{x}{n}\right)^n \geq 1-x \tag{6}$$

■

Using the binomial theorem, the expression $\left(1-\frac{x}{n}\right)^n$ can be expanded as follows.

$$\left(1-\frac{x}{n}\right)^n = \sum_{k=0}^{n}\binom{n}{k}\left(-\frac{x}{n}\right)^k = 1-x+\sum_{k=2}^{n}\binom{n}{k}\left(-\frac{x}{n}\right)^k \tag{7}$$

To complete the proof, we need to show that the alternating series $\sum_{k=2}^{n}\binom{n}{k}\left(-\frac{x}{n}\right)^k$ is greater than or equal 0. As the first term in the series is positive, we only need to show that all the terms have decreasing absolute values. In order to do so, we now show that the ratio between two consecutive terms is greater than 1.

$$\left|\frac{\binom{n}{k}\left(-\frac{x}{n}\right)^k}{\binom{n}{k+1}\left(-\frac{x}{n}\right)^{k+1}}\right| = \frac{\frac{n!}{k!\cdot(n-k)!}}{\frac{n!}{(k+1)!\cdot(n-k-1)!}\cdot\frac{x}{n}} = \frac{n\cdot(k+1)}{(n-k)\cdot x} \tag{8}$$

It is clear that the quantity at the right end side of Equation 8 is greater than 1, as $n\cdot(k+1)\geq n$ and $(n-k)\cdot x \leq n$. Using Theorem 2, we can conclude that

$$\left(1-\frac{\Pr\left(success\left([0,T]\right)\right)}{n}\right)^n \geq 1-\Pr\left(success\left([0,T]\right)\right) \tag{9}$$

Combining Equations 2, 5, and 9, we can write

$$1-\Pr\left(success\left([0,T],n\right)\right) \geq 1-\Pr\left(success\left([0,T]\right)\right) \tag{10}$$

Equation 1 follows directly from Equation 10.

In conclusion, Theorem 1 shows that, in theory, the proposed mechanism is effective in reducing the probability that the attacker will successfully discover currently used cryptographic keys in a given amount of time. In other words, it will take more time for the attacker to break the system. Experiments reported in the next section confirm this result.

In the following, we will refer to the *level of security* as a security metric to express how secure is a configuration with respect to the considered attacks. A security value can be assigned, based on the attacks coverage table, both to a single node and to a link, defined as a connection between communicating nodes. Node security is related

primarily to the physical layer (e.g., tamper resistant HW, protected external ROM), while subnet security depends on the security layer (e.g., cryptographic algorithm, key length, key agreement mechanisms); as previously discussed, both also depend on the reconfiguration mechanism itself, that is on time.

Assume that the set SEC of available cryptosystems is a totally ordered set: given s_1, $s_2 \in SEC$, there is an ordering relation between them, and $s_1 \leq s_2$ means that the cryptosystem s_1 is not more secure than the cryptosystem s_2. It is possible to have elements in SEC that are equivalent from the security point of view, adopting for instance the same algorithm but using different parameters (e.g. different keys).

Assume the sequence of the M configurations adopted by a node n is given by $\langle C_1(n), \ldots, C_M(n)\rangle$, and the sequence of time instants in which such configuration were activated is $\langle T_1(n), \ldots, T_M(n)\rangle$.

Let the configurations $C_i(p) = (im_i(p), api_i(p), s_i(p))$ and $C_i(q) = (im_i(q), api_i(q), s_i(q))$ be the i-th active configurations respectively on node p and q. Note that in order for the nodes to be able to communicate, they should either share the same security and API configurations, or they should be provided with a mechanism to always know what is the configuration currently used by other legitimate nodes. Let $T_i(p, q)$ identify the initial time instant when the status of p and q is such that they are able to communicate. In the following, we will refer to a $link$ as a directed edge (p, q) connecting two nodes involved in a communication, with packets traveling from p to q. A link configuration is defined as $C_i(p, q) = (C_i(p), C_i(q))$.

Let us refer to $SL_{(p,q)}(t)$ as the level of security, at time t, of a link (p, q). It is the level of security associated with the cryptosystem used to secure data flow from p to q, denoted with $s_i(p, q)$. With $SL_p(t)$ we identify the level of security of node p, depending on its physical configuration $im_i(p)$ and on time.

Definition 1 (Level of security of a link). *The level of security $SL_{(p,q)}(t)$ of a link (p, q), provided by $C_i(p, q)$ at time $t \in [T_i(p, q), T_{i+1}(p, q)]$, can be expressed as a function of the specific cryptosystem adopted $s_i(p, q)$ and the time elapsed since the current configuration was activated.*

$$SL_{(p,q)}(t) = f(s_i(p, q), t - T_i(p, q)) \tag{11}$$

Definition 2 (Level of security of a node). *The level of security $SL_p(t)$ of a node p can be expressed as a function of the specific firmware adopted $im_i(p)$ and the time elapsed since the current physical configuration was activated.*

$$SL_p(t) = f(im_i(p), t - T_i(p)) \tag{12}$$

Finally, we can define the Security Level associated to a configuration as:

Definition 3 (Level of security of the network). *Assuming that the network is partitioned in different subnets, each composed of nodes communicating with one another with a certain interface (security and application layer), the overall level of security of the network depends both on the security of nodes composing the network, and of the different subnets, other than on time.*

$$SL_{net}(t) = A \cdot \sum_{i=0}^{N-1} \sum_{j=0, j \neq i}^{N-1} \alpha_{ij} \cdot SL_{(i,j)}(t) \cdot x_{ij} + B \cdot \sum_{i=0}^{N-1} \beta_i \cdot SL_i(t) \tag{13}$$

where

- *A and B represent the relative importance of the set of links and the set of network nodes respectively, and satisfy the following constraint:* $A + B = 1$.
- *the* α_{ij} *constants represent link weights, and the* β_i *constants represent node weights and are useful to give more importance to critical nodes or portions of the network. They are subject to the following constraints:*

$$\begin{gathered} \sum_{i=0}^{N-1} \sum_{j=0, j \neq i}^{N-1} \alpha_{ij} = 1 \\ \sum_{i=0}^{N-1} \beta_i = 1 \end{gathered} \tag{14}$$

- *the* x_{ij} *variables represent the existence of links and are defined as follows:*

$$x_{ij} = \begin{cases} 1 \; if \; i \neq j \text{ and } \exists \text{ a link between node } i \text{ and } j \\ 0 \; if \; i \neq j \text{ and } \nexists \text{ a link between node } i \text{ and } j \end{cases} \tag{15}$$

4 WSN Reconfiguration: a Case Study

A WSN is an embedded network composed of a base station – able to perform multi-node data fusion and complex application logic, and often provided with a consistent source of energy – and several motes, which merely perform local processing on sensed data. Nodes communicate by exchanging messages over a radio channel: the base station sends queries to motes in order to sample physical variables (e.g., humidity), whereas motes simply reply to these queries by sending unicast messages to the base station.

Security is a fundamental concern in WSNs, as they are widely adopted in several critical application domains. Nevertheless, because of their peculiar features – constrained processing and storage capabilities, limited battery life, highly dynamic topology and mobility, frequent failures – providing security is not a straightforward task. The introduction of security mechanisms has a strong impact on performance and resource consumption, that often represent a limiting factor. For this reason, although the adoption of a complex cryptosystem (e.g., based on public key primitives) for all network activities could be desirable from a security point of view, it is not feasible in practice. The proposed reconfiguration approach is able to overcome these concerns, as it allows to maintain an acceptable level of security in the network by leveraging not only the intrinsic features of the adopted cryptosystems, but also other features, such as the physical configuration and the application interfaces, other than the reconfiguration mechanism itself. This way, the use of simpler cryptosystems for short periods of time can be preferable to the adoption of a single strong but computation-intensive cryptosystem.

In this discussion, we refer to TinyOS, the most commonly adopted operating system for WSNs. TinyOS applications and the OS itself are built by connecting components that represent functional building blocks, such as communication protocols, device drivers, or data analysis modules. During the default compilation process of TinyOS, these building blocks are converted into a monolithic, static binary, to enable code optimization and ensure a small memory footprint. This means that the OS and its applications' executables lack modularity, and it is not possible to dynamically replace a single component at runtime. Security mechanisms could be implemented either as independent TinyOS components or as different static libraries wired in the same component, whose functions are invoked by applications to ensure security requirements. Reconfiguration of both the security layer and the application interfaces could be easily achieved by including the implementation of all the available solutions into the firmware installed on the device, and activating the desired configuration through software switches and ad hoc protocols. Firmware reconfiguration can be performed by adopting node reprogramming techniques, that will be illustrated in details later. Two innovative approaches to reconfiguration are presented in the following subsections, along with some implementation details.

4.1 Security Layer Reconfiguration

Assume that, in order to enforce security, queries are signed by the base station for authentication purposes, and reply messages are encrypted for ensuring confidentiality and integrity. The security layer performing these operations can be designed to implement different cryptographic protocols, depending on the required security level and available resources. The basic idea of the proposed approach is to dynamically change the security layer, by switching between two or more different implementations. We assume that each node is provided with a pool of different cryptosystem implementations, which are identified by a unique ID.

To give a concrete example, we refer to the two cryptosystems presented in [28], based respectively on Elliptic Curve Cryptography (WM-ECC libraries) and Identity-based cryptographic techniques (TinyPairing libraries). WM-ECC [29] provides key agreement algorithms and digital signature that can be used to authenticate packets in the sensor network. It provides support for all the ECC operations and we used it to implement a hybrid cryptosystem [28] based on a public key function for ensuring authentication of the base station, and on a key agreement protocol for establishing a symmetric key, to be used for encryption/decryption of data packets sent by the motes. TinyPairing [30] is an open-source pairing-based cryptographic library for wireless sensors, providing an interesting solution to the key management problem, that still represents an open issue in WSN security research.

From a security point of view, the cryptosystem based on WM-ECC does not authenticate public keys, thus allowing man-in-the-middle attacks in the key exchange phase. Moreover, sensitive data is encrypted with a symmetric cipher, and this increases overall vulnerability of the network. Instead, TinyPairing adopts an asymmetric scheme and is much more secure in the initialization phase as it does not use a key exchange mechanism. As stated in Section 3.1, the intrinsic level of security of the two configurations can be represented by an attack coverage table, identifying, for each configuration,

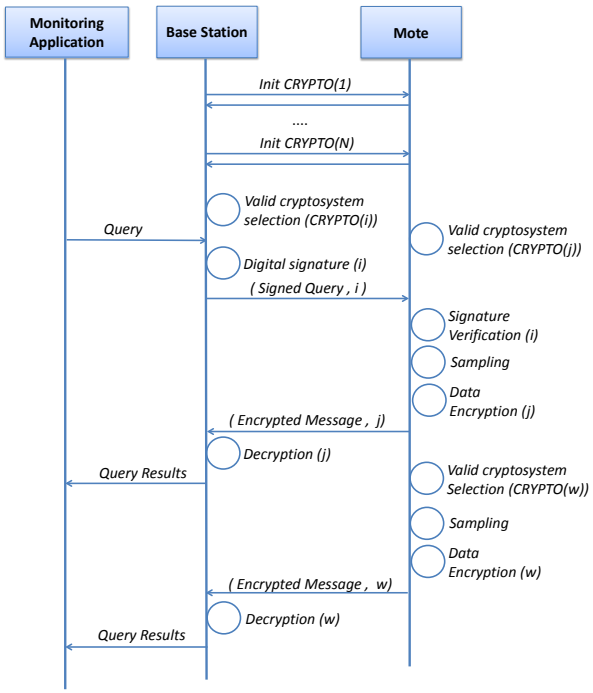


Fig. 2. Security protocol reconfiguration

what attacks it is able to thwart or, dually, what requirements it is able to satisfy. Table 2 shows an example of attack coverage table for the two cryptosystems (configurations) considered here. In this case, coverage is not defined as a binary property, but through a qualitative score capturing the level of protection provided by the configuration with respect to each considered attack.

Table 2. Attack Coverage Table for the considered cryptosystems

Configuration	*Man-in-the-middle*	*Eavesdropping*	*Brute force*	*Replay attack*
WM-ECC	non-auth key	yes	weak symm	no
TinyPairing	auth key	yes	asymm	no

In the simplest reconfiguration scenario, each node can decide independently when to update, and an identifier of the cryptosystem used to encrypt a message is encoded in the message itself, so that each receiving node, sharing the same reconfiguration strategy, is able to properly handle it.

Fig. 2 illustrates a typical scenario for security protocol reconfiguration. In the INIT phase, the base station and the motes agree on the parameters of N different cryptosystems. The details of the initialization phase depend on the specific cryptosystems (the parameters can be public points for an ECC based cryptosystem, or system parameters for an identity based [28]). Initialization should be performed in a secure environment, either in the pre-deployment phase or later. After initializing the N available cryptosystems, each node can independently choose the valid cryptosystem to adopt for performing cryptographic operations in the current validity interval. In particular, the base station choses the cryptosystem it will use to digitally sign the outgoing queries and ensure authentication (CRYPTO(i) in figure). Any mote receiving the query message will use the cryptosystem whose ID is included in the message itself to verify the signature. Similarly, after verifying the signature, any mote encrypts data using the locally selected cryptosystem (CRYPTO(j) and CRYPTO(w) in figure), and the base station will use the ID included in the reply messages to decrypt them.

As illustrated in Fig. 2, the parameters of the N cryptosystems (i.e., the cryptographic keys) could be either preloaded on all network nodes in the INIT phase, or dynamically determined in each reconfiguration phase according to available key agreement mechanisms. All the cryptographic keys can be stored in each node for the entire lifetime of the network, and they can be used as master keys for generating new keys.

As discussed in Section 3, the introduction of reconfiguration mechanisms impacts a system's performance by introducing some overhead depending on the mechanism itself and the particular architectural level it is applied to. Each reconfiguration solution that encompasses switching among different implementations of the same functionality is characterized by an unavoidable increase in memory storage, at least to include the different available versions and the additional mechanisms to implement the reconfiguration itself. Referring to the application security reconfiguration mechanism proposed in this section, each node will be loaded with an application image including the binaries of all the available security libraries and their initialization parameters, along with the software switches needed to dynamically select different security primitives. Clearly, this could represent a problem for constrained devices such as sensor nodes, that are typically equipped with a small flash memory and RAM, therefore a more prudent design of security libraries should be devised in order to save as much memory as possible. As for energy consumption and required CPU computational effort, the proposed solution does not affect them, since cryptographic operations belonging to different libraries are activated by simple software switches. Moreover, there is no latency to swap from a cryptosystem to another and we do not need to stop the monitoring application during the reconfiguration.

Let us now take a look at possible security weaknesses of this strategy: an attacker who is aware of the message format may try to manipulate some fields of query and data packets, such as those coding the cryptosystem ID and its parameters, so that nodes are no longer able to communicate. As for query messages, their payload is signed with the base station's private key, so that, if any field is altered during transmission, the signature verification at mote's side will not succeed, and the message will be discarded. This aspect of the protocol could be exploited to execute a denial of service attack, with motes not able to verify the authenticity of queries and thus refusing to provide the

required data. To detect this type of attack, a timeout is set by the base station every time a query is sent. If no reply is received before the timer expires, the query is sent again to cope with possible message losses. If no reply is received after a few attempts, an alert is raised. The cryptosystem ID is also encoded in each response message, as it is necessary to decrypt the message. An attacker could alter it as messages are not authenticated, but then the base station will not be able to decrypt them, and will discard them. This situation may cause the loss of some response messages. However, as a typical sensor network is composed of many redundant motes, this situation is not critical.

4.2 Physical Layer Reconfiguration

Several existing approaches for sensor network reprogramming perform a *full-image replacement*, consisting in completely replacing the image of the application running on a node. Deluge [31] is a reliable data dissemination protocol for propagating large data objects (larger than a node's memory) from one or more source nodes to many other nodes over a multi-hop network. As Dutta *et al.* pointed out in [32], this approach is unsafe and too battery-consuming. We implemented a different approach to remotely reconfigure each node in the network. We decoupled the reconfiguration mechanisms from the components to enforce the new configuration according to a scheduling policy.

To this aim, we designed a reconfiguration application by augmenting several components of the Deluge framework. In particular, we implemented new reconfiguration functionalities to enable a single node to swap to a new image that was previously preloaded on its storage. The reconfiguration application is defined by wiring new components specifically designed to manage external reconfiguration commands, and components designed to manage the images loaded on the node storage. The proposed reconfiguration application consists of three main components, namely (i) a bootloader component, (ii) a reprogramming component, and (iii) a management component.

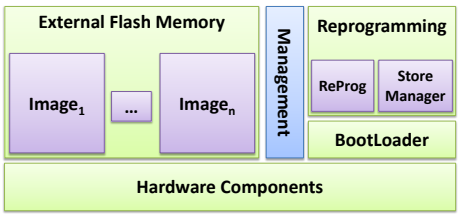


Fig. 3. Reconfiguration Application components

The *bootloader component* is a persistent layer in the architecture, which can enforce the chosen reconfiguration mechanisms. This component is intended for TinyOS and provides needed functionalities to program the node with an already stored program image. The parameters passed to this component are specified in the external command and indicate the location of the binary in the external flash memory to program the node's microcontroller. When reprogramming is requested, the bootloader will erase

the program flash and write the new binary to it. On completion, it jumps to the first instruction of the new application.

The *reprogramming component* is the core of the *reconfiguration application*. In our implementation, it accepts commands from the base station, but can be extended to implement a decentralized reconfiguration approach. This component is built by connecting two primary subcomponents: the *ReProg* and the *StorageManager*. The ReProg component is an extension of the NetProg component of Deluge T2. It handles a reprogramming request from the network by providing a dedicated API to initialize a reconfiguration process. When a node wants to perform a reconfiguration, it only has to invoke this API by specifying the name of the new binary in the flash memory to load. Subsequently, the ReProg sets the environment variables needed by the bootloader component and reboots the node. The StorageManager component deals with image name resolution, mapping names of program images to their respective physical addresses in the external flash memory.

The *management component* has a master (base station) and a mote side. It is used to initialize the mote and deploy different images. Usually this operation is done in a secure environment and it is accessible only during the initialization. The master-side *management component* has been derived from the *tos-deluge* application of the Deluge T2 Framework, and it is called mote-manager. This component allows to inject one or more images into the mote by writing directly into nodes' external flash memory volumes. It is also possible to erase a volume and ping the status of a mote to get information about already injected images.

Finally, the reconfiguration application runs on a workstation connected to the base station, which implements the reprogramming scheduler. As discussed in the next section, the reprogramming frequency and the new configuration to load can be chosen to balance overhead and attack probability.

Clearly, physical layer reconfiguration introduces a greater overhead than security layer reconfiguration, as it is based on full image replacement. As previously discussed, due to resources limitations, a node's memory cannot be preloaded with many different application images. Consider that the monitoring application we implemented occupies about 16 KB of ROM and 700 Bytes of RAM on a telosB platform, equipped with a 48 KB ROM memory and a 10 KB RAM.

The proposed solution introduces considerable advantages in terms of overall performance with respect to WSN reprogramming approaches based on code dissemination. In fact, the reconfiguration time is now not dependent on the image size and the network topology as the images are not sent over the network but preloaded via a serial interface. This approach avoids any security risk in the dissemination phase, and reduces the battery consumption as it only needs the introduction of a simple command message to swap from an image to another one. As the available application images are all loaded on the node's external memory, the swapping latency is considerably reduced with respect to the code dissemination case. We experimented a reduction of one order of magnitude with respect to the original Deluge approach: from 50 seconds to send a 40Kb image implementing a monitoring application secured with WM-ECC, to about 6 seconds to perform the swap.

One known drawback of this approach is the need to stop the monitoring application and any ongoing query in order to swap to another image. Nevertheless, as previously discussed, any available approach that is based on full image replacement presents even worst issues. A possible solution to mitigate this problem could be that of delaying the reconfiguration operation if the node is involved in some communication, or pausing an ongoing communication until reconfiguration is completed. It is up to network administrators to decide frequencies and strategies for node reconfiguration. As for possible security weaknesses of this solution, some attacks can be considered undermining this reconfiguration mechanism. First of all, preloading nodes with all images exposes them to physical compromise. We can assume that, in presence of strict security requirements, nodes are equipped with tamper-resistant packages so that they cannot be compromised. Moreover, an attacker may try to replay control packets sent by the base station and containing a reconfiguration command, in order to control communication or simply perform a denial of service attack by forcing motes to continuously swap images. This risk can be prevented by introducing a sequence number for reconfiguration commands and proper channel encryption.

5 MTD Evaluation

In order to evaluate the effectiveness of security-driven reconfiguration – even under adverse conditions – we assume that an attacker is able to understand when the adopted cryptosystem changes and what type of cryptosystem is used at each time (e.g., by observing control messages sent over the network by the base station in the node reconfiguration strategy, or control flags found in data packets in the protocol reconfiguration strategy).

Many types of cryptographic attacks can be considered. In our case, an attacker can only observe encrypted packets traveling on the network and containing information about sensed data, and can perform a brute force attack on captured packets by systematically testing every possible key for the current (known) cryptosystem – assuming he is able to determine when the attack is successful. In the *worst case* (for the defender), the attacker knows the encryption algorithm and the key length associated with the algorithm, therefore he can systematically try all possible keys of that length. In the *intermediate case*, the attacker knows the encryption algorithm but does not know the key length associated with it, thus he systematically tries all possible keys for a given set of key lengths. In the *best case*, the attacker does not know anything about the adopted cryptosystem, thus he tries all possible keys for a given set of key lengths and a given set of cryptosystems.

We evaluated our approach with respect to the cryptosystems described in [28], whose characteristics are summarized in Table 3.

The WM_ECC_sk and WM_ECC_rc5 cryptosystems are both based on the WM-ECC library, used to execute key exchange and digital signature operations. They both perform symmetric encryption using respectively a Skipjack cipher with a 80 bit key and an RC5 cipher with a key of 160 bits. The TinyPairing cryptosystems is based on TinyPairing and uses a 208 bit key. In Table 3, the time needed to test a single key is reported for each cryptosystem, along with the maximum attack time, that is the time

necessary to test all the possible keys. The reported execution times (third column) refer to the execution of the decryption operation on TelosB devices, equipped with a 4.15 MHz MSP430 microcontroller, a CC2420 radio chip, a 10 KB internal RAM, and a 48 KB program flash memory.

The maximum attack times reported in the fourth column of Table 3 have been computed analytically based on the measured time needed to perform a single decryption operation. It is important to point out that these attack times are significantly high due to the nature of the attacks we considered. In practice, attacks may be more sophisticated and efficient than brute force attacks. However, this does not affect the validity of the proposed MTD approach as we are interested in illustrating how the probability of successfully completing an attack decreases, compared to a static configuration scenario.

Table 3. Characteristics of cryptosystems

Cryptosystem	*key lenght (bits)*	*time (ms)*	*max attack time (ms)*
WM_ECC_sk	80	0.001251	1.5123E+21
WM_ECC_rc5	160	0.001221	1.7845E+45
TinyPairing	208	13.019531	5.3560E+63

We carried out our experiments considering both worst and intermediate cases, and analyzed the cumulative distribution function (cdf) of the *attack time*. In both cases, we simulated an attacker sequentially exploring the key space. We considered an observation interval as long as the attack time of the most complex cryptosystem, TinyPairing, and validity intervals of decreasing length. A validity interval is the time interval in which a single system configuration is active. At the end of the i-th validity interval, the new configuration to activate should be chosen based on a specific strategy (e.g., related to security or battery consumption requirements) in order to maximize reconfiguration benefits. However, for the sake of simplicity, the results shown in this section have been obtained by performing random choices among the available cryptosystems at each validity interval. In particular, during an observation interval, we randomly generated 1,000 different sequences of valid cryptosystems and recorded the time of successful attacks to build the cdf. Clearly, the sequence length depends on the chosen validity interval.

In the first experiment, we chose three validity interval lengths in such a way to be comparable to the maximum attack times of the three different cryptosystems. The resulting attack time's cdf in the worst case is shown in Fig. 4: the labels in the figure – note that the x-axis is on a logarithmic scale – identify three inflection points in the middle of the maximum attack times of each cryptosystem. These correspond to the maximum values of the attack time probability distribution functions (pdf) for each cryptosystem.

When analyzing the chart, a seemingly counterintuitive behavior can be identified: when considering smaller validity intervals the attacker seems to benefit.

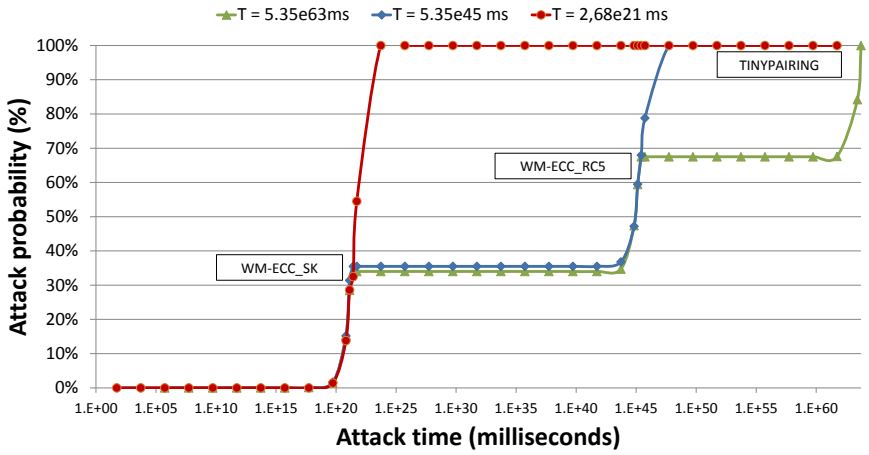


Fig. 4. Worst case attack time cdf for large validity intervals

The chart can be explained as follows. The WM_ECC_sk cryptosystem can be certainly broken in 1.5123E+21 ms (worst case for WM_ECC_sk) as shown in Table 3. This means that, if randomly selecting one cryptosystem among the 3 available, and choosing a validity interval greater than this threshold, the cryptosystem will always be broken. As each cryptosystem has a 33% probability of being selected at next reconfiguration time – based on our assumptions – in 33% of cases the system will be broken. Similar considerations can be made for the other two cryptosystems, explaining the other inflection points.

As illustrated in Fig. 5, when reducing the length of the validity interval – with validity intervals larger than the maximum attack time of the weakest cryptosystem – the attack time increases, with the percentage of successful attacks reducing dramatically. The same behavior is highlighted in Fig. 6, which shows how the probability of completing a successful attack within a time t varies as the length of the validity interval changes: as soon as the validity interval drops below the maximum attack time of the weakest cryptosystem, the rate at which probability decreases becomes higher.

Similar results can be obtained when reconfiguration is performed by selecting an equivalent cryptosystem that uses different parameters (i.e different keys). Fig. 7(a) shows the attack time's cdf in the worst case when reconfiguration is performed by switching among three cryptosystems that implement the WM-ECC library with the Skipjack cipher, but have different keys. When reducing the validity interval, the probability of successfully completing an attack significantly decrease as the intrinsic security level is restored every time a new key is adopted. For comparison purposes, Fig. 7(b) shows the attack time's cdf when three different cryptosystems are used. As expected, increased diversity results in a lower probability of attack.

Fig. 8 compares the attack time's cdf for the intermediate and the worst cases, under the assumption that the attacker performs a brute force attack using the set of key lengths in Table 4. The validity interval of 5,36E+45 milliseconds is long enough to break

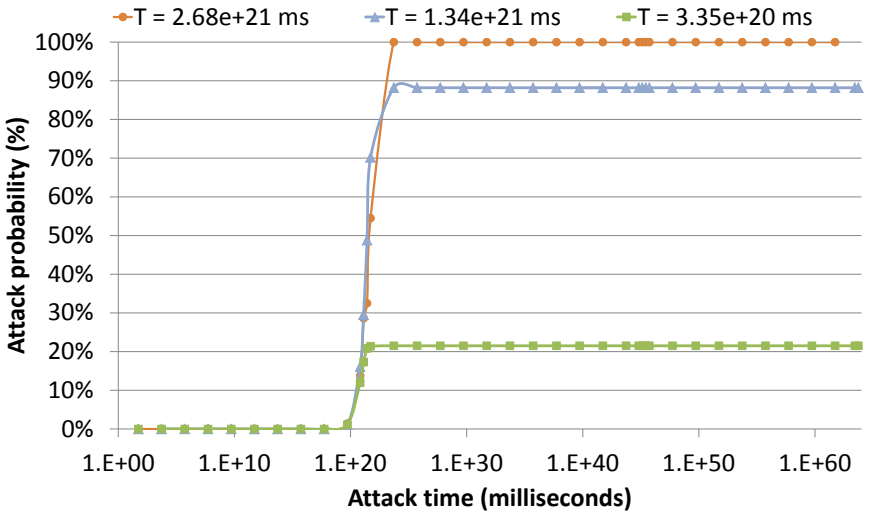


Fig. 5. Worst case attack time cdf

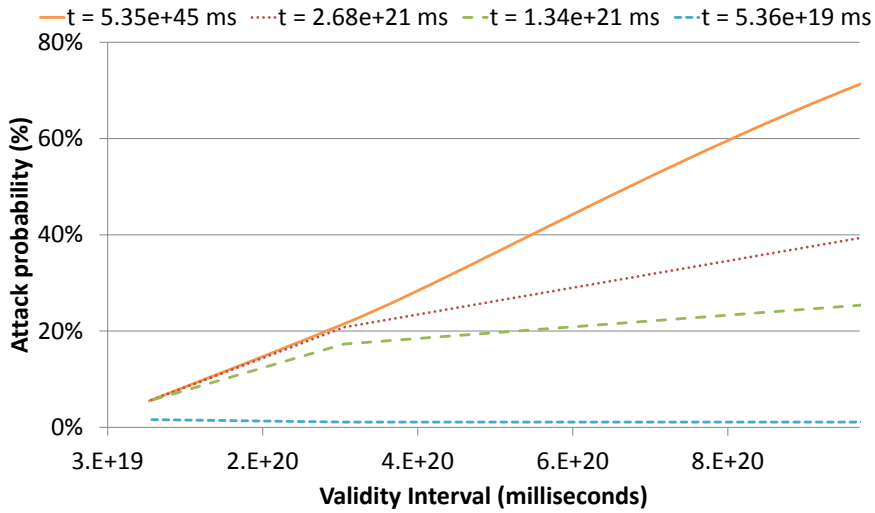
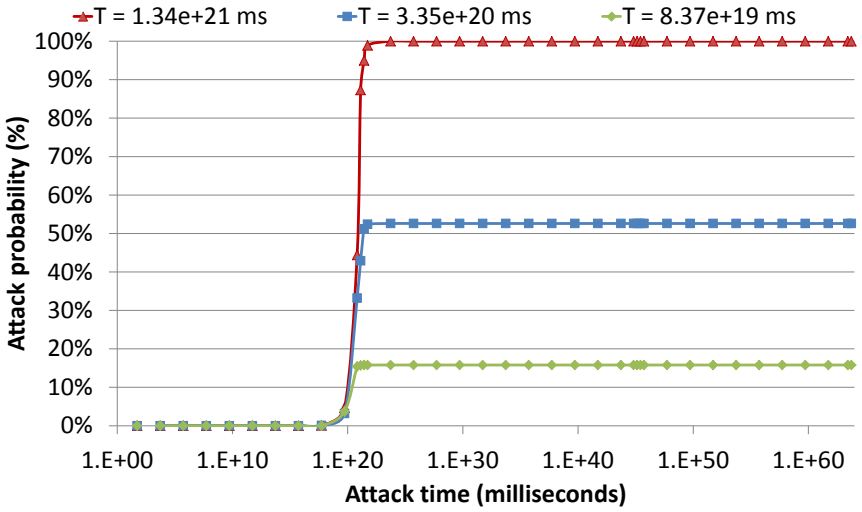
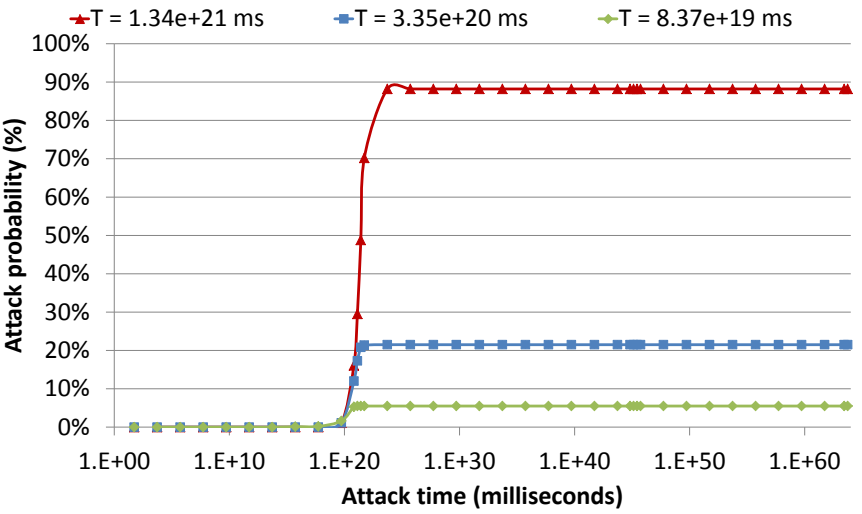


Fig. 6. Probability of successful attack

both WM_ECC_sk and WM_ECC_rc5. As shown, the attacker's success probability is smaller in the intermediate case. Clearly, when the attacker's uncertainty about the used cryptosystem is higher, more key lengths will be tested, making the proposed approach even more effective.



(a) Same cryptosystem with different keys



(b) Three different cryptosystems

Fig. 7. Worst case attack time cdf

6 Conclusions

In this paper, we have proposed an MTD approach for protecting resource-constrained distribute devices. The proposed approach is based on fine-grained *reconfiguration* at different architectural layers. Changing configuration or system parameters to augment security is an intuitive principle, but there is still a lack of metrics to evaluate the security

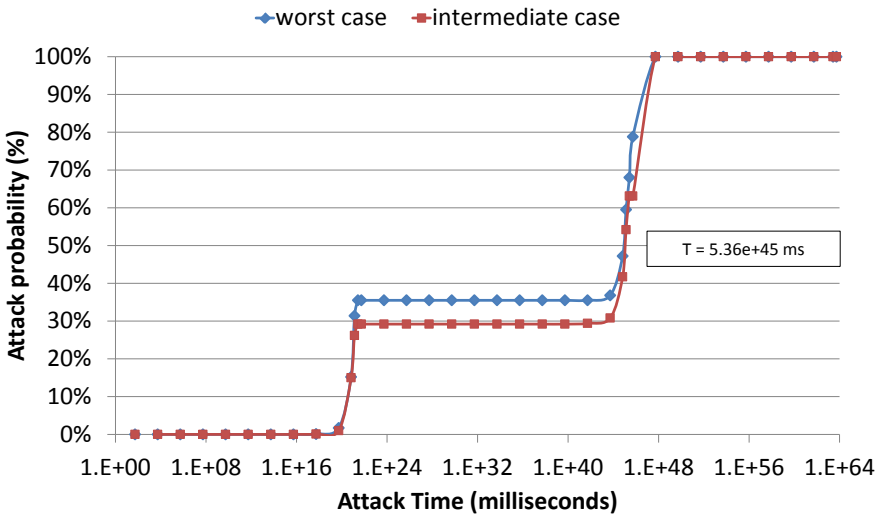


Fig. 8. Worst case vs. intermediate case

Table 4. Key lengths set

Cryptosystem	*key len (bits)*	*time(ms)*
WM_ECC_sk	[80]	[0.001251]
WM_ECC_rc5	[120,160]	[0.001120,0.001221]
TinyPairing	[180,208]	[11.023211,13.019531]

level of a system and quantify the benefits of reconfiguration. We have introduced two innovative MTD mechanisms to reconfigure the network, and experimentally showed that the proposed mechanisms are effective in increasing the complexity for the attacker to successfully complete an attack. In the near future, we plan to work on different ways to extend and generalize this approach. Indeed, we are already working on a formal model of reconfiguration. Furthermore, we plan to define mechanisms to automatically enforce reconfiguration strategies based on external events or on the system's state.

References

1. Executive Office of the President, National Science and Technology Council: Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program. http://www.whitehouse.gov/ (December 2011)
2. Jajodia, S., Ghosh, A.K., Subrahmanian, V.S., Swarup, V., Wang, C., Wang, X.S., eds.: Moving Target Defense II: Application of Game Theory and Adversarial Modeling. 1st edn. Volume 100 of Advances in Information Security. Springer (2013)

3. Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S., eds.: Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats. 1st edn. Volume 54 of Advances in Information Security. Springer (2011)
4. Wang, Q., Zhu, Y., Cheng, L.: Reprogramming wireless sensor networks: Challenges and approaches. IEEE Networks **20**(3) (May 2006) 48–55
5. Casola, V., De Benedictis, A., Albanese, M.: A moving target defense approach for protecting resource-constrained distributed devices. In: Proceedings of the 14th IEEE International Conference on Information Reuse and Integration (IEEE IRI 2013), San Francisco, CA, USA (August 2013)
6. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Transactions on Software Engineering **37**(3) (May 2011) 371–386
7. Evans, D., Nguyen-Tuong, A., Knight, J.C.: Effectiveness of Moving Target Defenses. In: Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats. Springer (2011) 29–48
8. Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering code-injection attacks with instruction-set randomization. In: Proceedings of the 10th ACM conference on Computer and communications security. CCS '03, New York, NY, USA, ACM (2003) 272–280
9. Cristian Cadar, Periklis Akritidis, M.C.J.P.M., Castro, M.: Data randomization. Technical report, Microsoft Research (2008)
10. Jackson, T., Salamat, B., Homescu, A., Manivannan, K., Wagner, G., Gal, A., Brunthaler, S., Wimmer, C., Franz, M.: Compiler-Generated Software Diversity. In: Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats. Springer (2011) 77–98
11. Antonatos, S., Akritidis, P., Markatos, E.P., Anagnostakis, K.G.: Defending against hitlist worms using network address space randomization. Computer Networks **51**(12) (August 2007) 3471–3490
12. Atighetchi, M., Pal, P., Webber, F., Jones, C.: Adaptive use of network-centric mechanisms in cyber-defense. In: Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2003). (May 2003) 183–192
13. Kewley, D., Fink, R., Lowry, J., Dean, M.: Dynamic approaches to thwart adversary intelligence gathering. In: Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX 2011). Volume 1., Anaheim, CA, USA (June 2011) 176–185
14. Antonatos, S., Akritidis, P., Markatos, E., Anagnostakis, K.: Defending against hitlist worms using network address space randomization. Computer Networks **51**(12) (2007) 3471 – 3490
15. Jafarian, J.H., Al-Shaer, E., Duan, Q.: Openflow random host mutation: transparent moving target defense using software defined networking. In: Proceedings of the first workshop on Hot topics in software defined networks. HotSDN '12, New York, NY, USA, ACM (2012) 127–132
16. Albanese, M., De Benedictis, A., Jajodia, S., Sun, K.: A moving target defense mechanism for MANETs based on identity virtualization. In: Proceedings of the First IEEE Conference on Communications and Network Security (IEEE CNS 2013), Washington, DC, USA (October 2013)
17. Casola, V., Mazzeo, A., Mazzocca, N., Vittorini, V.: A policy-based methodology for security evaluation: A security metric for public key infrastructures. Journal of Computer Security **15**(2) (2007) 197–229
18. Common criteria project: Common criteria for information technology security evaluation 2.1. Technical report, US NIST (1999)
19. Trusted computer system evaluation criteria. Technical Report DoD 5200.28-STD, US Department Of Defense (1985)
20. Li, X., Parker, T.P., Xu, S.: A stochastic model for quantitative security analyses of networked systems. IEEE Trans. Dependable Sec. Comput. **8**(1) (2011) 28–43

21. Barth, A., Rubinstein, B.I.P., Sundararajan, M., Mitchell, J.C., Song, D., Bartlett, P.L.: A learning-based approach to reactive security. IEEE Trans. Dependable Sec. Comput. **9**(4) (2012) 482–493
22. Ahmed, M.S., Al-Shaer, E., Khan, L.: A novel quantitative approach for measuring network security. In: INFOCOM. (2008) 1957–1965
23. Pamula, J., Jajodia, S., Ammann, P., Swarup, V.: A weakest-adversary security metric for network configuration security analysis. In: QoP. (2006) 31–38
24. Mell, P., Scarfone, K., Romanosky, S., Mell, P., Scarfone, K., Romanosky, S., Gutierrez, C.M., Director, W.J.: The common vulnerability scoring system (cvss) and its applicability to federal agency systems (2007)
25. Casola, V., Mazzeo, A., Mazzocca, N., Vittorini, V.: A policy-based methodology for security evaluation: A security metric for public key infrastructures. Journal of Computer Security **15**(2) (April 2007) 197–229
26. Casola, V., Preziosi, R., Rak, M., Troiano, L.: A reference model for security level evaluation: Policy and fuzzy techniques. Journal of Universal Computer Science **11**(1) (2005) 150–174
27. Foley, S.N., Fitzgerald, W., Bistarelli, S., OSullivan, B., Foghl, M.: Principles of Secure Network Configuration: Towards a Formal Basis for Self-configuration. Volume 4268 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006)
28. Casola, V., Benedictis, A.D., Drago, A., Mazzocca, N.: Analysis and comparison of security protocols in wireless sensor networks. In: Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems Workshops (SRDSW 2011), Madrid, Spain (October 2011) 52–56
29. Wang, H., Sheng, B., Tan, C., Li, Q.: WM-ECC: An elliptic curve cryptography suite on sensor motes. Technical Report WMCS-2007-11, College of William and Mary (October 2007)
30. Xiong, X., Wong, D.S., , Deng, X.: TinyPairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks. In: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2010). (April 2010)
31. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), Baltimore, MD, USA (2004) 81–94
32. Dutta, P.K., Hui, J.W., Chu, D.C., Culler, D.E.: Securing the deluge network programming system. In: Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN 2006). (April 2006) 326–333