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

Valentina Casola and Alessandra De Benedictis
Department of Electrical Engineering and Information Technology
University of Naples Federico II
Naples, Italy
{casolav,alessandra.debenedictis}@unina.it

Massimiliano Albanese
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. 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.

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 [9, 13, 14]. As stated in [9], 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 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. 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 most such efforts have primarily 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 between multiple lightweight cryptosystems. Several reconfiguration mechanisms have been proposed for WSNs

*The work presented in this paper is supported in part by the Army Research Office under award number W911NF-12-1-0448, and by the Office of Naval Research under award number N00014-12-1-0461.

IEEE IRI 2013, August 14-16, 2013, San Francisco, California, USA
978-1-4799-1050-2/13/$31.00 ©2013 IEEE

[18], 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 is organized as follows. Section 2 discusses the benefits of an MTD approach to reconfiguration in embedded networks, and introduces the main reconfigurable architectural layers. Section 3 illustrates two innovative reconfiguration mechanisms for WSNs, wheras Section 4 reports experimental results. Finally, some concluding remarks are given in Section 5.

2 MTD Approach to Node Security

Moving Target Defense (MTD) [13, 14] provides a way to make more difficult for an attacker to exploit a vulnerable system. The idea 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. According to the definition in [16], 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.

As suggested in [8], MTD approaches (also referred to as *diversity techniques*) may be applied both at the application level or at a lower level (e.g., code location in memory). 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. 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 [10, 12].

Several higher-level MTD approaches have also been proposed, and most of them are aimed at thwarting the attacker's reconnaissance effort [2, 3, 15]. 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. The MTD defense mechanism proposed in [1] 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.

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.

In this paper, we propose an MTD framework for reconfiguring resource-constrained devices at different levels, with the reconfiguration granularity chosen at runtime based on current requirements. 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 embedded in 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 runtime. 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.

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.

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.

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.

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 that quantifies the effort an attacker needs to break it [5, 6]. 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 Figure 1, using reconfiguration, we can prevent the security level from falling below a certain

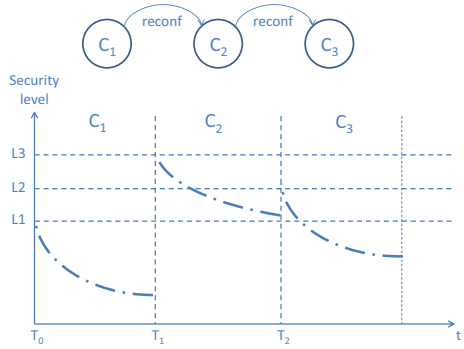


Figure 1. Reconfigurations and security level

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 reconfiguration, 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.*

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

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.*

The proof of Theorem 1 is quite straightforward, but we omit it for reasons of space.

3 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.

3.1 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.

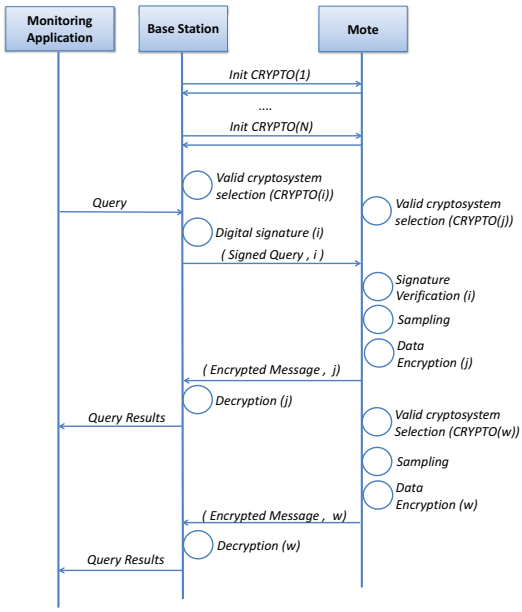


Figure 2. Security protocol reconfiguration

To give a concrete example, we refer to the two cryptosystems presented in [4], based respectively on Elliptic Curve Cryptography (WM-ECC libraries) and Identity-based cryptographic techniques (TinyPairing libraries). WM-ECC [17] 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 [4] 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 [19] 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.

In the simplest 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.

Figure 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 [4]). 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 Figure 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. The main advantages of this solution relate to improved overall performance. In fact, there is no latency to swap from a cryptosystem to another and we do not need to stop the monitoring application during the reconfiguration; even battery consumption is not affected by this solution.

As for the security 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.

3.2 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 [11] 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 [7], 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.

The *bootloader component* is a persistent layer in the architecture, which can enforce the chosen reconfiguration mechanisms. This component is intended for TinyOS and it 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.

The proposed solution introduces considerable advantages in terms of security and 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. Furthermore, this approach avoids any security risk in the dissemination and reduces the battery consumption as the messages sent are simply commands to swap from an image to another one. The swapping latency is considerably reduced, too. 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. The only drawback is that we need to stop the monitoring application and any query being executed in order to swap to another cryptosystem. However, any approach based on full image replacement presents similar issues. Furthermore, due to storage limitations, we can only preload a limited number of images on board.

We can now consider possible attacks aimed at undermining this reconfiguration mechanism. 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.

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

Table 1. Characteristics of cryptosystems

4 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 [4], whose characteristics are summarized in Table 1. 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.

It is important to point out that attack times resulting from our simulations 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.

We carried out our simulations 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.

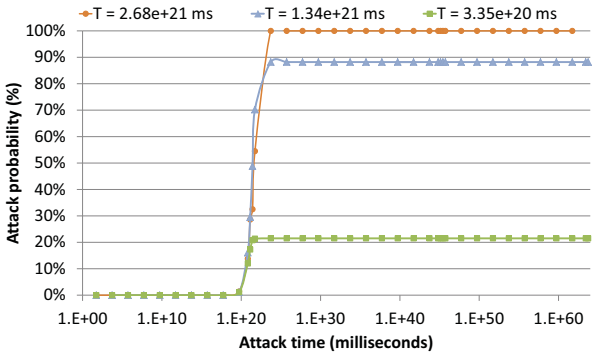


Figure 3. Worst case attack time cdf

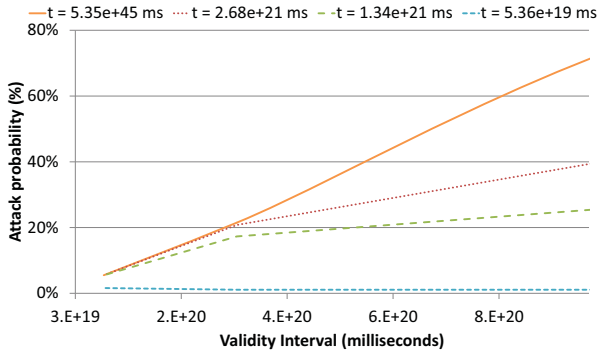
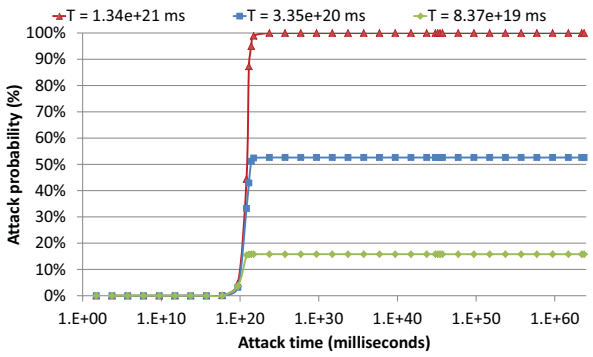


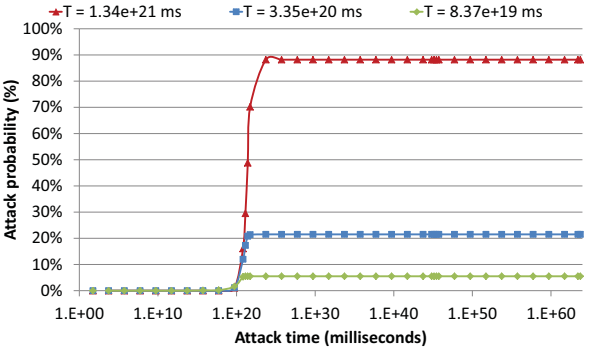
Figure 4. Probability of successful attack

Figure 3 shows the attack time's cdf in the worst case: as expected, 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 Figure 4, 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). Figure 5(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, Figure 5(b) shows the attack time's cdf when three different cryptosystems are used. As expected, increased diversity



(a) Same cryptosystem with different keys



(b) Three different cryptosystems

Figure 5. Worst case attack time cdf

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]

Table 2. Key lengths set

results in a lower probability of attack.

Figure 6 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 2. The validity interval of 5,36E+45 milliseconds is long enough to break 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.

5 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

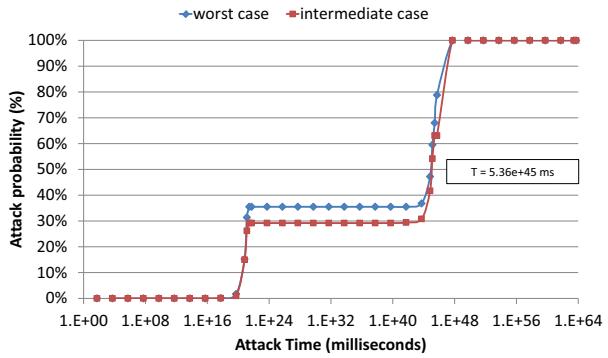


Figure 6. Worst case vs. intermediate case

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 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] M. Albanese, A. D. Benedictis, S. Jajodia, and K. Sun. A moving target defense mechanism for manets based on identity virtualization. In *Proceedings of the First IEEE Conference on Communications and Network Security (CNS 2013)*, Washington, DC, USA, October 2013.

[2] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471–3490, August 2007.

[3] M. Atighetchi, P. Pal, F. Webber, and C. Jones. 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)*, pages 183–192, May 2003.

[4] V. Casola, A. D. Benedictis, A. Drago, and N. Mazzocca. Analysis and comparison of security protocols in wireless sensor networks. In *Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems Workshops (SRDSW 2011)*, pages 52–56, Madrid, Spain, October 2011.

[5] V. Casola, A. Mazzeo, N. Mazzocca, and V. Vittorini. A policy-based methodology for security evaluation: A security metric for public key infrastructures. *Journal of Computer Security*, 15(2):197–229, April 2007.

[6] V. Casola, R. Preziosi, M. Rak, and L. Troiano. A reference model for security level evaluation: Policy and fuzzy techniques. *Journal of Universal Computer Science*, 11(1):150–174, 2005.

[7] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the deluge network programming system. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN 2006)*, pages 326–333, April 2006.

[8] D. Evans, A. Nguyen-Tuong, and J. C. Knight. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, chapter Effectiveness of Moving Target Defenses, pages 29–48. Springer, 2011.

[9] 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.

[10] S. Forrest, A. Somayaji, and D. H. Ackley. Building diverse computer systems. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, pages 67–72, 1997.

[11] J. W. Hui and D. Culler. 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)*, pages 81–94, Baltimore, MD, USA, 2004.

[12] T. Jackson, B. Salamat, A. Homescu, K. Manivannan, G. Wagner, A. Gal, S. Brunthaler, C. Wimmer, and M. Franz. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, chapter Compiler-Generated Software Diversity, pages 77–98. Springer, 2011.

[13] S. Jajodia, A. K. Ghosh, V. S. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang, editors. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, volume 100 of *Advances in Information Security*. Springer, 1st edition, 2013.

[14] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, editors. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, volume 54 of *Advances in Information Security*. Springer, 1st edition, 2011.

[15] D. Kewley, R. Fink, J. Lowry, and M. Dean. Dynamic approaches to thwart adversary intelligence gathering. In *Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX 2011)*, volume 1, pages 176–185, Anaheim, CA, USA, June 2011.

[16] P. K. Manadhata and J. M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386, May 2011.

[17] H. Wang, B. Sheng, C. Tan, and Q. Li. WM-ECC: An elliptic curve cryptography suite on sensor motes. Technical Report WMCS-2007-11, College of William and Mary, October 2007.

[18] Q. Wang, Y. Zhu, and L. Cheng. Reprogramming wireless sensor networks: Challenges and approaches. *IEEE Networks*, 20(3):48–55, May 2006.

[19] X. Xiong, D. S. Wong, , and X. Deng. 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.