

A Specification-based Intrusion Detection Engine for Infrastructure-less Networks

Christoforos Panos¹, Christos Xenakis², Platon Kotzias², Ioannis Stavrakakis¹

¹Department of Informatics & Telecommunications, University of Athens, Greece

{cpanos, ioannis}@di.uoa.gr

²Department of Digital Systems, University of Piraeus, Greece

{xenakis, platon}@unipi.gr

Abstract

The proliferation of mobile computing devices has enabled the utilization of infrastructure-less networking as commercial solutions. However, the distributed and cooperative nature of routing in such networks makes them vulnerable to a variety of attacks. This paper proposes a host-based monitoring mechanism, called SIDE that safeguards the operation of the AODV routing protocol. SIDE encompasses two complementary functionalities: (i) a specification-based detection engine for the AODV routing protocol, and (ii) a remote attestation procedure that ensures the integrity of a running SIDE instance. The proposed mechanism operates on a trusted computing platform that provides hardware-based root of trust and cryptographic acceleration, used by the remote attestation procedure, as well as protection against runtime attacks. A key advantage of the proposed mechanism is its ability to effectively detect both known and unknown attacks, in real time. Performance analysis shows that attacks are resolved with high detection accuracy, even under conditions of high network volatility. Moreover, SIDE induces the least amount of control packet overhead in comparison with a number of other proposed IDS schemes.

Keywords: MANET, IDS, AODV, detection engine, attestation.

1 Introduction

Infrastructure-less networks comprise a wide range of networking paradigms such as mesh networks, mobile ad hoc networks (MANETs), vehicular ad hoc networks (VANETs), delay tolerant networks (DTN), opportunistic and sensor networks, as well as various overlay networks. A common characteristic of these networks is the absence of any fixed architectural component such as routers, access points, etc., supporting and serving dynamic topologies and behaviors. These unique properties, empowered by the proliferation of mobile devices (i.e., smartphones, tablets, etc.) and the advent of ad-hoc networking standards, such as Wi-Fi direct [1], enable the materialization of infrastructure-less networks for providing communication and cooperation solutions, such as the extension of networking environments (i.e., cellular networks, personal or corporate wireless networks, etc.) in areas where network coverage is limited [3] (i.e., metropolitan areas, indoor environments, etc.).

A widely accepted implementation of an infrastructure-less network is based on a dynamic and adaptive routing protocol, named ad hoc on demand distance vector (AODV) [2], which, initially, was designed for MANETs and later has been adopted by DTN [4], opportunistic [5], mesh [6], and sensor [7] networks. AODV operates with the assumption that all participating nodes are well-behaved, and thus, it does not include any security mechanism. Considering also the deployment characteristics of infrastructure-less networks (i.e., wireless shared access, dynamic topologies, cooperative routing, etc.), it can be realized that AODV faces a wide set of security threats [11]. More specifically, any malicious network node may easily exploit critical protocol fields such as *hop count*, *sequence numbers*, *source and destination address*, etc., causing a variety of attacks, such as route disruption, resource consumption, denial of services, etc. [9].

Since the protection of the protocol's fields and functionality is not possible by default, an effective way to address these inherent vulnerabilities is through the deployment of a detection mechanism. However, the design of an intrusion detection system (IDS) for AODV has been proven a challenging task, considering the limitations of the existing IDS [8][23] (i.e., analyzed in sect. 2.2 of this paper). The majority of them capture, store, and, subsequently, process the whole traffic (i.e., control and payload) within the radio range of a monitoring node, in order to collect as much audit data as possible and then assess the behavior of the neighboring nodes. Consequently, monitoring nodes bear additional computational and storage burdens, while energy consumption is increased. In addition, during the collection of audit data, malicious activities are not detected. Finally, in cases of high nodes' mobility or continuous changes in network topology, the collected audit data might lead to inconclusive or erroneous assessments, resulting in false positives/negatives.

The limitations and weaknesses of current IDSs may be addressed by a host-based IDS that monitors the behavior of its own host node. A host-based IDS alleviates the need for collecting audit data that may be malicious, incomplete, or outdated, providing an accurate and real time view of the host node's protocol operations. Thus, malicious behaviors can be detected immediately, with low false positives/negatives, and without the associated overheads of audit data collection. However, such an approach has been unfeasible in the past, mainly, because of the fact that a host-based IDS operating on a malicious node, could not be considered as trusted. The emergence of trusted computing [20] may address this uncertainty and make host-based IDS a viable security solution for infrastructure-less networks. Trusted computing provides hardware-based root of trust, accompanied by a set of primitive functions that propagates trust from hardware to the application software. At the core of this technology resides the process of remote attestation with which a computer can prove the integrity of a platform (e.g., hardware and software) to a remote party [38].

This paper proposes a novel host-based monitoring mechanism, called SIDE (i.e., Specification-based Intrusion Detection), which relies on trusted computing in order to provide a resilient, specification-based IDS. More specifically, each network node implements an instance of SIDE, which unlike existing IDSs, is responsible for monitoring its own host node. This approach enables SIDE’s detection engine to monitor local information and ascertain an accurate view of protocol operations, in real time. SIDE’s detection engine is based on a comprehensive *set of specifications* that defines the legitimate functionality of the AODV protocol. As a result, any malicious activity (i.e., known or unknown) that violates the legitimate functionality of AODV can be identified. To defend against malicious host nodes that may attempt to modify or even disable SIDE, the proposed mechanism encompasses a *remote attestation procedure* that verifies the integrity of running SIDE instances in the network. Moreover, SIDE operates on a *trusted computing platform* that provides hardware-based root of trust and cryptographic acceleration, used by the remote attestation procedure, as well as protection against runtime attacks. The proposed mechanism utilizes a TrustZone [42] enabled ARM processor, which constitutes a trusted computing platform included in the vast majority of mobile and embedded devices. The performance of SIDE is evaluated through an extensive set of simulations. The numerical results show that SIDE resolves attacks in real time with high detection accuracy, while imposing limited overheads in the operation of AODV.

The rest of this paper is organized as follows. Section 2 analyzes the functionality of the AODV routing protocol; briefly evaluates existing security schemes that have been proposed for AODV; and provides an analysis of remote attestation techniques. In section 3 the proposed mechanism is introduced and its functionality is elaborated. In section 4, we perform an in-depth evaluation of SIDE, which includes: (i) an outline of its advantages over previously proposed detection engines; (ii) a security evaluation of its robustness against a variety of attacks; (iii) the computational cost and memory requirements, and, (iv) a comparative evaluation of its performance based on simulations. Finally, section 5 contains the conclusions.

2 Background

In this section, we first provide an overview of the AODV protocol’s functionality. This overview covers only the most critical aspects of the protocol’s operations, since a more throughout analysis of AODV exists in [2]. In section 2.2, we provide an evaluation of several security solutions that have been proposed for AODV. A comprehensive analysis of all the related literature requires an extensive review, which is outside the scope of this paper. Instead, we have selected a representative set of security solutions that covers the majority of utilized security mechanisms and encompasses: (i) extensions to the AODV protocol that

incorporate cryptography and (ii) intrusion detection mechanisms that use either anomaly-based or specification-based detection. Finally, in section 2.3, we evaluate existing remote attestation procedures.

2.1 Overview of the AODV routing protocol

AODV is an on demand routing protocol, which maintains routes as long they are needed by source nodes. It is scalable and offers low processing, memory, and communication overheads to the underlying network. It utilizes three control messages to achieve route discovery: route request (RREQ), route reply (RREP), and route error (RERR). It also provides an optional fourth control message (i.e., Hello message), which is used for preserving connectivity between neighboring nodes. When a node wishes the establishment of a route, it initiates a route discovery process by broadcasting a RREQ message that includes the: *source IP address*, *source sequence number*, *destination IP address*, *destination sequence number*, *RREQ id* (i.e., an incremented identifier), and *hop count field*. Each RREQ message is, uniquely, identified by the pair of *source IP address* and *RREQ id*. The intermediate nodes that receive the RREQ may either reply to it (i.e., possess an updated route to the destination) or forward it (i.e., do not possess a route to the destination and the time to live (TTL) field is greater than one). In case that multiple copies of the same RREQ are received by an intermediate node, the duplicates are discarded. The destination node or an intermediate node that has a fresh route to the destination replies to a RREQ, by generating an RREP message that contains the: *source IP address*, *source sequence number*, *destination IP address*, *destination sequence number* (i.e., an increasing counter denoting the most recent route), *lifetime field* (i.e., indicates the time for which the route is considered valid), and *hop count field* (i.e., denotes the distance in hops from the source to the destination). Intermediate nodes receiving the RREP update their routing tables, only, if the *destination sequence number* in the message is higher from the stored value in their routing tables, or the *destination sequence numbers* are equal, but the *hop count field* in the RREP is smaller than the stored value. If a link breaks, an intermediate node initiates a local repair mechanism attempting to discover a new route to the destination by transmitting a RREQ message. If the repair mechanism fails to discover a route, the node generates a RERR message that includes the *IP addresses* and the last known *destination sequence numbers* of the unreachable destinations, informing the receiving nodes that they should restart the routing discovery process, if they want to communicate with them.

A node offers connectivity information by broadcasting local Hello messages, if this feature is enabled. Every time-period of *hello interval*, the node broadcasts a Hello message, which contains the: *destination IP address*, *destination sequence number*, *lifetime field*, and *hop count field*. The *lifetime field* is assigned the value $allowed\ hello\ loss * hello\ interval$,

while the *hop count* is set equal to zero. The *allowed hello loss* parameter is used by network administrators to determine the time frame (i.e., in multiples of the *hello interval*), where the routes are considered valid. Nodes perceive connectivity by listening to the packets transmitted by their neighbors. If a node does not receive any packet from a neighbor for a time period greater than $allowed\ hello\ loss * hello\ interval$, it assumes that the link to this node is currently lost.

2.2 Related work

The Secure AODV (SAODV) [36] is one of the first security mechanisms proposed for the AODV protocol. It constitutes a security-enhanced version of AODV that aims at protecting the routing messages of AODV, through the use of cryptography. It uses digital signatures to authenticate the non-mutable fields of messages and hash chains to authenticate the hop-count field, in both RREQ and RREP messages. However, this functionality requires extensive modifications to the original AODV protocol, raising compatibility issues. Furthermore, the authors assume that the key pairs used for the production of digital signatures cannot be compromised and thus, do not incorporate any self-protecting mechanisms. Finally, SAODV is unable to protect against blackhole, wormhole, rushing, and DoS attacks [17].

Trying to limit the required modifications and attempting to protect from a wider set of threats, the majority of AODV security mechanisms uses detection techniques. Particularly, [29] proposes an anomaly-based engine that employs machine learning to generate a normal profile and relies on principal component analysis (PCA) [47] for detecting denial of service (DoS) attacks. However, the generated normal profile is static, including only the network conditions of the time that it was created. Therefore, in case of network changes during time, the engine considers them as results of malicious behaviors, presenting high rates of false positives. To address this, the use of dynamically updated normal profiles has been proposed in [25][26][30], where monitoring data during a period of time in which no malicious behavior was detected, is used to update the normal profile. The first [25] of these adaptive solutions uses PCA for resolving malicious behaviors; the second [26] utilizes a support vector machine classifier (SVM) [27] for detecting sinking attacks (i.e., nodes that do not cooperate in routing and forwarding); and the third one [30] relies on statistical analysis of malicious RREQ flooding (MRF1) for detecting DoS attacks. Although the use of dynamic profiles may reduce the rate of false positives in volatile networks; on the other hand, it is prone to false negatives, if within a monitoring time-period the engine fails to detect a malicious behavior, while an attack(s) takes place. In this case, the carried attack(s) becomes part of the normal profile, remaining undetected. Moreover, in order to limit the associated overheads of capturing, storing, and processing audit data, the aforementioned engines are

configured to monitor, only, a limited set of features, which enable the detection of a restricted set of possible attacks [25][26][29].

Another approach that attempts to address the limitation of static normal profiles is through the use of dynamic thresholds. In [28], the authors have proposed a two-stage anomaly-based detection. During the first stage, a node is considered a potential threat if its observed behavior results in the highest statistical deviation from a pre-computed normal profile; while a threshold is generated (i.e., used during the second stage) by averaging the scores of all the observed nodes. In the second stage, the observed behavior of nodes is compared for statistical deviation from the generated threshold. Using this approach, any periodic symptom of suspicious behavior, caused mainly by network volatility, may lead to an improperly generated threshold, and consequently, to either false positives (i.e., if the generated threshold is too low) or false negatives (i.e., if the threshold is set too high).

In [37], a distributed cooperative mechanism (DCM) is proposed to resolve blackhole attacks, by monitoring data packets transmitted by neighboring nodes. If a node has not routed any data packets during a fixed time-threshold, then the monitoring node will transmit a “test packet” through the suspicious node, destined for another cooperating detection node. If the later receives the “test packet,” then the suspicious node is legitimate, otherwise it is considered malicious. The primary disadvantage of this scheme is that malicious nodes may attempt to exploit this mechanism, by analyzing the duration of time before a malicious node is detected (i.e., estimate the threshold value), and subsequently, the routing of at least one packet within this time-frame.

The inconstant detection accuracy of anomaly-based detection, which is extrapolated by network volatility, can be addressed using specification-based detection. However, the existing specification engines for AODV present some serious design limitations. Specially, the engines presented in [13][15][16] rely on distributed monitoring, by setting the nodes into promiscuous mode and observing the exchanged RREQ and RREP control messages. However, an attacker is able to generate and then forward forged RERR messages, disrupting in this way the routing process, while remaining undetected. Moreover, high nodes’ mobility has an impact on the delay of RREP messages, resulting in false positives. To address such limitations, both works presented in [14] and [18] propose two engines relying on more comprehensive sets of specifications, based on the known AODV attacks. However, attacks that do not violate these specifications remain undetected. Having recognized this, the authors of [14] have proposed the operation of a supplementary anomaly-based engine, which on the other hand eliminates all the advantages of employing specification-based detection.

Regardless of the detection approach (i.e., specification or anomaly-based), all of the aforementioned schemes do not take into account any security vulnerabilities that may arise in relation to the proposed security mechanism itself. A security mechanism, such as an IDS,

should not only be capable of detecting malicious behavior, but also avoid introducing new vulnerabilities and be resilient to attacks that target the mechanism itself. Adversaries trying to avoid detection may attempt to target the employed IDS, aiming at hindering its operation, disabling it or tampering its functionality. Furthermore, a compromised IDS can be used to launch additional attacks, such as providing erroneous detection results in order to falsely accuse legitimate nodes as malicious.

In our previous work [10], we introduced the concept of combining host-based monitoring and specification-based detection as a means for addressing the aforementioned limitations of present IDSs, as well as proposed a set of specifications that covers the functionality of the 802.11 MAC protocol. In this paper, we significantly expand that work by: (i) proposing a comprehensive set of specifications that covers the critical functionality of the AODV protocol, (ii) enhancing the resilience of SIDE through the utilization of a trusted computing platform (i.e., TrustZone), (iii) proposing a remote attestation procedure that facilitates trust between monitoring entities, (iv) providing a security evaluation, identifying possible attacks and vulnerabilities of SIDE and outlining how these are addressed, and (v) carrying out a performance evaluation, in order to evaluate the induced overheads associated with SIDE.

2.3 Remote attestation

Remote attestation may ensure the integrity of a software object such as a detection engine, by employing software or hardware techniques. In software techniques, the attester issues a challenge to the target node, requesting the verification of software hosted by the later. This challenge comes in the form of a verification procedure incorporating obfuscation or code optimality techniques [22], which should be executed within a predetermined time-period. Obfuscation attempts to transform the verification procedure into an equivalent, which is hard to be reversed engineered. Code optimality, on the other hand, relies on the assumption that the verification procedure cannot be further optimized in order to be executed faster, and thus, any attempt to reverse-engineer it will add noticeable delays. The target node hashes the memory location in which the verified software resides at and completes the procedure by returning the hashed results to the attester. The limitations of such a technique are summarized below: (i) it cannot be deployed in infrastructure-less networks (especially DTN), since any network-induced latency may lead to false positives; (ii) it cannot detect runtime attacks, such as buffer overflow; and (iii) it induces latency by disabling interrupts and blocking the execution of other software/applications during the verification procedure [32].

Hardware-based attestation can be achieved based on trusted computing platforms. One such platform relies on a special hardware component, called trust platform module (TPM), to

establish a chain of trust through the basic input/output system (BIOS) to the operating system (OS), where load-time integrity is ensured using hash values stored in a sealed storage memory within TPM. Currently, several manufacturers such as Broadcom and Toshiba produce TPM microcontrollers, based on the latest TPM specifications [20]. However, this approach presents some significant limitations: (i) it increases the cost of devices due to the additional hardware; (ii) it cannot be deployed on legacy devices; (iii) it does not protect against runtime attacks; (iv) it relies on the assumption that a TPM cannot be tampered; and (v) in case of a TPM compromise, the trusted hardware must be physically replaced. Dynamic software integrity mechanisms [38][40] have also been designed to complement hardware-based attestation, supporting runtime verification. They employ additional integrity measurements, at runtime, by monitoring data structures, system calls, memory stack, etc., but this binary analysis induces significant computational overhead, resulting in an execution slowdown of 3000% to 5000%.

ARM Corporation offers an alternative trusted computing platform, known as TrustZone [42], suitable for ARM processor cores and system on chip (SoC) components. It provides two virtual processing cores with different privileges and a strictly controlled communication interface, enabling the creation of two distinct execution environments, encapsulated by hardware. It does not include a remote attestation procedure, but it offers all the required services to create one (i.e., secure storage, cryptographic primitives, etc.). Other vendors, such as Intel and AMD also provide similar platforms (i.e., Intel's TXT, and AMD's secure execution environment), but ARM equips the majority of mobile and embedded computing devices.

3 The proposed mechanism

In this section, we analyze the proposed mechanism's operation. In particular, section 3.1 provides an architectural overview of all the components utilized by each network node in order to deploy the proposed mechanism. In section 3.2, we propose a comprehensive set of specifications that defines the legitimate functionality of the AODV protocol. The detection engine relies on this set to resolve if the behavior of the host node is legitimate or malicious. It is important to mention that although we focus on the AODV protocol, the proposed mechanism can be also applied to any infrastructure-less routing protocol, such as the Dynamic Source Routing (DSR) [48] and the Destination Sequence Distance Vector (DSDV) [49], by establishing a corresponding set of specifications. Finally, in section 3.3, we propose a remote attestation procedure that enables each instance of SIDE to attest its valid and un-tampered operation to its neighboring entities.

3.1 General architecture

Each network node, where the proposed mechanism can be applied, should consist of a TrustZone enabled SoC, which provides two virtual cores that execute two OSs: the un-trusted and the trusted (see Figure 1). The un-trusted OS holds the user’s applications and data, including the instance of the AODV protocol that enables the host-node to communicate with other network nodes. Its kernel performs the basic functions such as application execution, memory management, device driver installation and maintenance, etc. Software executed in the un-trusted OS may also perform malicious actions that require escalated privileges (i.e., “root” rights). On the other hand, the trusted OS provides a protected execution environment for SIDE (i.e., both the detection engine and the remote attestation procedure), which is not accessible by the un-trusted OS (i.e., any code or data contained in SIDE cannot be inspected or modified by applications/software executed within the un-trusted OS).

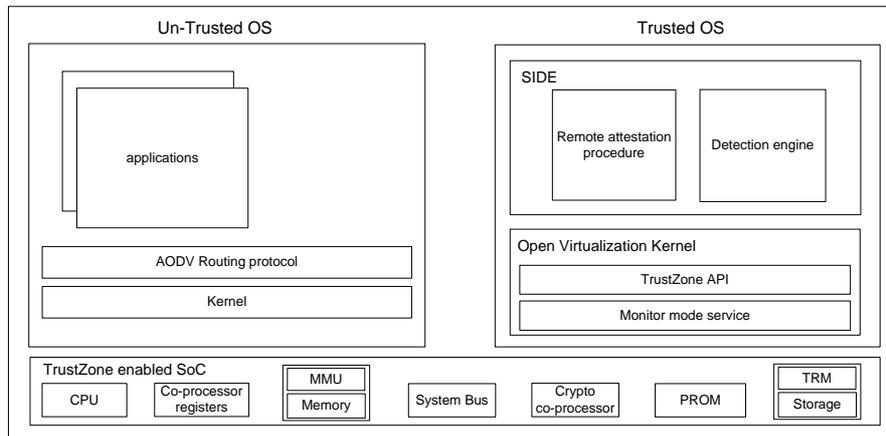


Figure 1: Architecture of a network node operating SIDE

The TrustZone SoC provides both the underlying hardware in which a node operates on, as well as hardware support for enforcing access control of hardware resources between the trusted and un-trusted OS. It encompasses the central processing unit (CPU), a set of co-processor registers, the volatile memory (i.e., RAM), the memory management unit (MMU), the system’s bus, a cryptographic co-processor, the one-time programmable memory (i.e., PROM), the storage unit, and the trusted region mapping unit (TRM). MMU, the system’s bus, and the TRM include a non-secure (NS) flag that instructs the CPU whether the underlying resources are used by the un-trusted or the trusted OS, respectively. For example, for every RAM storage read/write request, the CPU checks the respective NS field for the particular memory address (i.e., MMU responsible for mapping virtual addresses to physical addresses) or partition (i.e, TRMs responsible for partitioning storage) and decides whether it belongs to the un-trusted or trusted OS. Access to the system bus (i.e., responsible for interconnecting all of the SoC’s hardware components) is controlled in a similar fashion. The

co-processor registers, MMU, the cryptographic co-processor, PROM, TRM and NS flags are neither visible, nor accessible from the un-trusted OS.

PROM is used to store an SHA256 hash of a pre-distributed public key (i.e., NA_{PuK}), which is stored in the host node's storage. Since this hash value cannot be tampered or overwritten, it can be used to infallibly verify the public key NA_{PuK} , which in turn can be used to authenticate nodes during the remote attestation procedure (see section 3.3 for more detail). The crypto co-processor provides basic crypto functionality, including key generation, encryption and decryption, using industry standard encryption algorithms such as RSA, AES, SHA2, etc. The goal behind this is to alleviate the CPU from computationally intensive cryptographic calculations, providing silicon that is specifically designed for this functionality. This approach increases the speed of cryptographic algorithms (i.e., throughput), while limiting their respective resources and battery consumption, making them feasible for resource constrained devices.

The trusted OS is implemented using the open virtualization kernel [43], (i.e., the first open source implementation of a trusted OS kernel), which provides the basic OS functionality (i.e., task, storage, and memory management), the monitor mode service, and the TrustZone API [44]. The monitor mode service handles context switching between the two virtual processor cores (i.e. trusted and un-trusted), facilitated by the co-processor registers that are used to store the state of the currently executing virtual core. This functionality is similar to a traditional OS context switch, ensuring that the state of the world that the CPU is leaving is safely saved, and the state of the world the CPU is switching to is correctly restored. Prior to the execution of SIDE, the monitor mode service will always perform a switch to the trusted OS (i.e., the trusted execution environment), and thus, the subsequent execution of SIDE will be isolated from the un-trusted OS though the aforementioned hardware restrictions, imposed by the TrustZone enabled CPU. The TrustZone API provides a standardized software interface to the TrustZone security services (i.e., cryptographic implementations using the cryptographic co-processor), which are employed by the remote attestation procedure of SIDE.

SIDE consists of a remote attestation procedure, which provides a framework for installing trust between SIDE entities, as well as an engine that monitors AODV operation, indicating any malicious behavior - activity. The remote attestation procedure (elaborated further in sect. 3.3) utilizes the TrustZone API's cryptographic primitives to verify software integrity on a target node, as well as to provide authentication and confidentiality between the communicating nodes. On the other hand, the detection engine (elaborated further in sect. 3.2) monitors the host node's AODV protocol operations in real time, and resolves whether some AODV operation constitutes a malicious behavior, based on a comprehensive set of specifications that accurately define the legitimate operation of AODV.

3.2 The detection engine

The set of specifications utilized by the detection engine to resolve malicious behavior is expressed through the use of a finite state machine (FSM). FSM states are characterized as legitimate or malignant, and a transition from one state to another is triggered by the node's operation or actions. The engine monitors the protocol's execution and maps it to the appropriate state (i.e., legitimate or malignant). States that have not been specified are also considered as malignant. The developed specifications are divided into three sets, based on the host-node's communication condition: (a) idle, (b) transmitting, and (c) receiving states. Nevertheless, the detection engine can be expressed as one inclusive FSM. In the following sections, the FSM states of the proposed specification-based detection engine are presented and analyzed.

3.2.1 Idle specifications

The first set of specifications is presented only for the sake of completeness, since in this condition the protocol is awaiting either for a transmission or reception of packets, and thus, it does not include any final state designating a malicious behavior. The engine is initialized at state S_0 and begins monitoring the host node for the following conditions: (i) a new packet is ready for transmission, (ii) an incoming data packet has arrived, or (iii) the reception of a control message. In the first two cases, the engine moves to states S_1 (see sect. 3.2.2) and S_2 (see sect. 3.2.3), respectively; while in the third case it moves to state S_3 (see sect. 3.2.3). A host node may receive three types of control messages: (i) a RREQ message, (i.e., as part of a route discovery process) and the engine moves to state S_4 (see sect. 3.2.3), (ii) a RREP message (i.e., as a reply to a route discovery) and the engine moves to S_5 (see sect. 3.2.3), or (iii) a RERR message (i.e., as a consequence of a link breakage) and the engine moves to S_6 (see sect. 3.2.2). Finally, if the host node has enabled the Hello message exchange (i.e., connectivity information), it may either generate a Hello message, where the engine monitors its generation (see sect. 3.2.2, state S_7), or receive one from a neighboring node, where the engine monitors the routing table update process (see sect. 3.2.3, state S_8).

3.2.2 Transmitter specifications

In case that the host node has a packet to transmit to another node (i.e., state S_1), the engine moves to state S_9 , (if the node has already an active route to the destination); otherwise, it moves to S_{10} . In state S_9 , the protocol retrieves from the routing table the route to the destination and attempts to transmit the packet (i.e., the engine moves to state S_{11} – see Figure 2). If the transmission is successful, the engine moves to the initial state S_0 ; otherwise, if a local link breakage occurs, AODV initiates the local repair mechanism attempting to discover a new route to the destination by transmitting a RREQ message, and the engine moves to state

S₁₀. In case that the local repair is successful and the packet is transmitted, the engine returns to S₁₁; otherwise, the protocol generates a RERR control message and the engine goes to S₆ (see Figure 5).

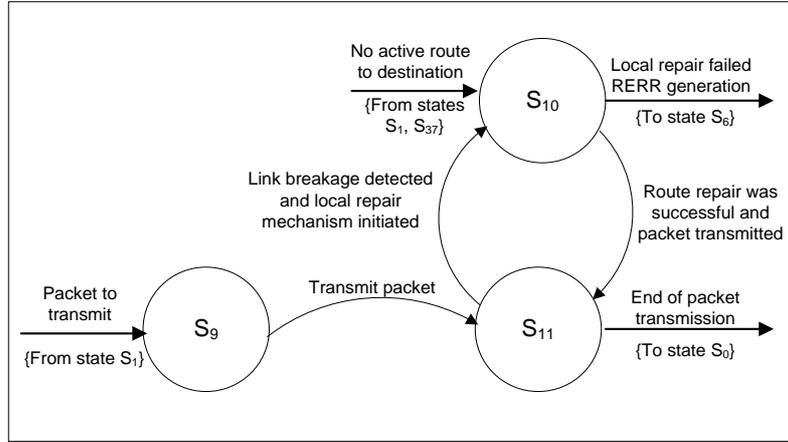


Figure 2: Transmission of a data packet having a route to the destination

In S₁₀ (i.e., no active route to the destination exists – see Figure 3), AODV prepares a RREQ control message and the engine moves to state S₁₂, where the following message fields are inspected: (i) the *destination sequence number* field should be equal to the last one stored in the routing table, and if no value is stored the *unknown sequence number flag* must be set up; (ii) the *originator sequence number* field must be set up equal to the host's *sequence number* incremented by one; (iii) the *RREQ id* field should be also incremented by one; iv) the *originator IP address* field should be equal to the *source address* of the host node; and v) the *hop count* field should be set equal to zero. If any mismatch occurs, the engine moves to the final state S₁₃; otherwise, the host node broadcasts the RREQ and the engine moves to state S₁₄. If AODV produces more RREQs per second than the *RREQ rate limit*, then the engine moves to the final state S₁₅, indicating a flooding attack. At state S₁₄, AODV awaits for a RREP message containing an active route to the destination. In case that the host node receives a RREP before the *net traversal* timer expires, the engine moves to state S₁₆ and the host node transmits the packet. After the transmission of the packet, the engine moves to state S₁₁ (see Figure 2). Otherwise, if the timer expires before the reception of a RREP, the engine moves to state S₁₇, the protocol initiates the backoff mechanism and then the engine moves to state S₁₈ (see Figure 4). If the node attempts to initiate the backoff mechanism before the timer expires, the engine moves to the final state S₁₅.

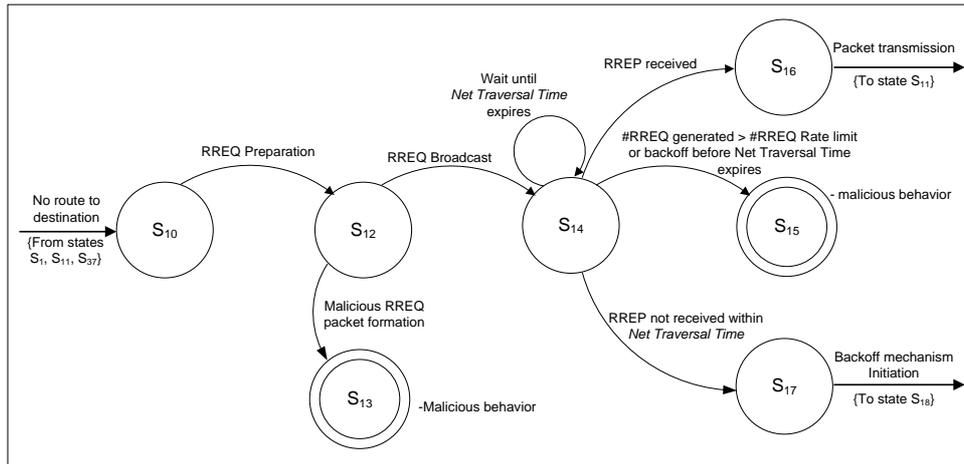


Figure 3: Transmission of a data packet without a route to the destination

The backoff mechanism, which is initiated in cases that the host node does not receive a RREP within a *net traversal time* period, manages how the host node generates RREQs, protecting the network from RREQ flooding. As illustrated in Figure 4, the monitoring phase of this mechanism begins with state S_{18} , in which AODV makes the first RREQ rebroadcast and the engine moves to state S_{19} . At this state, the host node is expected to wait for twice the previous *net traversal time*, before attempting to rebroadcast the RREQ. If the host node transmits it before the timer expires, the engine moves to the final state S_{20} , designating a *rushing attack*. If a RREP is received before the timer expires, the engine moves to state S_{21} and the host node transmits the packet. After the transmission of the packet, the engine moves to state S_{11} (see Figure 2). Otherwise, (i.e., the host node does not receive any RREP message after *RREQ retries* attempts within a *TTL max* timeframe), it stops the broadcasting of RREQs and drops the packet await for transmission. If it does not drop it, the engine moves to the final state S_{23} (i.e., malicious behavior). Finally, if the route request was initiated by a local repair (i.e., states S_{10} and S_{37} in Figure 2 and Figure 7, respectively), the protocol must generate a RERR message and the engine moves to state S_6 , else, it returns to the initial state S_0 .

Based on the AODV specifications [2], a RERR message is generated if one of the following occurs:

- The host node, while transmitting data, detects a link breakage for the next hop of an active route and the route repair attempt has failed (see state S_{10} , Figure 2).
- The host node receives a data packet destined to a node for which it does not have an active route and the initiated local repair mechanism has failed (see state S_{10} , Figure 7).
- The host node receives a RERR from a neighbor for one or more active routes.

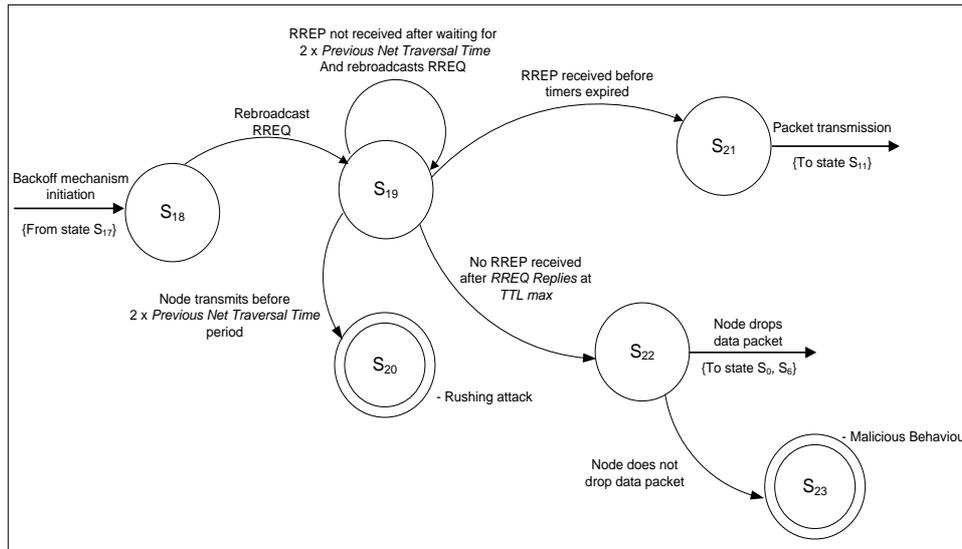


Figure 4: Backoff mechanism

In case that one of the above conditions occurs, the detection engine moves to state S_6 (see Figure 5), where AODV updates the affected entries of the host node's routing table and provides a list of the unreachable destinations. For case (a) and (c) this list consists of the unreachable neighbors as well as the destination nodes in the local routing table of the host that use them as next hop; while in case (b) the list includes only the destination of the data packet that cannot be delivered. After these, the engine moves to state S_{24} and validates the modifications to the routing table's field, allowing only the following: (i) for every unreachable destination, the *destination sequence number* of its routing entry (if such exists and is valid), in case (a) and (b) is incremented by one, while in case (c) it is copied from the received RERR message; (ii) the *route entry* is marked as invalid; and (iii) the *lifetime field* is updated to current time plus the *delete period*. If any deviation from the above takes place, then the engine moves to the final state S_{25} ; otherwise, after the generation of the RERR messages, it moves to state S_{26} . At this state, the engine checks the following RERR message fields: i) the *destination IP address* and the *destination sequence number* should match the corresponding fields of the unreachable destination included in the routing table; and ii) the *no delete flag* should be set, only, if the node has initiated the local repair mechanism. If any deviation takes place or the rate of generated RERR messages is greater than *RERR rate limit*, (i.e., a parameter included in AODV), the engine moves to the final state S_{27} ; otherwise, the host node broadcasts the RERR and the engine moves to state S_{28} . After the completion of the transmission, the engine returns to the initial state S_0 .

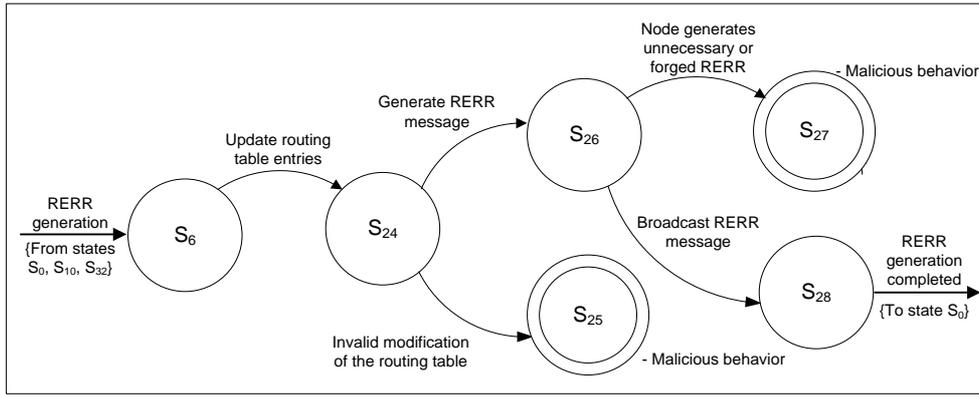


Figure 5: Generation of a RERR

The generation of a Hello message is a periodic process that takes place every *hello interval*. In S_7 (see Figure 6), the engine monitors the *hello interval* and when the timer expires, it moves to state S_{29} . The generation of a message before the timer's expiration, will lead to the final state S_{30} , designating malicious behavior. At state S_{29} , the engine validates the following fields of the Hello message: (i) the *destination IP address* and *destination sequence number* should be equal to the IP address and latest *sequence number* of the host node; (ii) the *hop count* have to be equal to 0; and (iii) the *lifetime* must be set equal to the product *allowed hello loss * hello interval*. If any of these fields holds an invalid value, then the engine moves to the final state S_{30} ; otherwise, it moves to S_{31} and monitors the transmission of the newly created message. In case that the host node drops the message (i.e., instead of broadcasting it), the engine moves to the final state S_{32} .

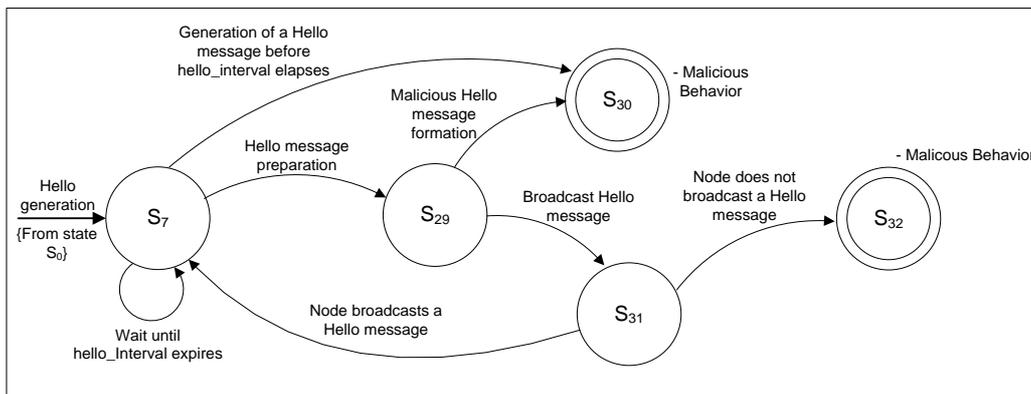


Figure 6: Transmission of a Hello message

3.2.3 Receiver specifications

In case of data reception (i.e., data packet or control message), the engine based on the packet's type moves from the idle state S_0 to either S_2 (i.e., data packet) or S_3 (i.e., control message). At state S_2 (see Figure 7), the engine checks if the host node is the final destination of the received data packet, and moves to state S_{33} ; otherwise (i.e., if the host node is an

intermediate destination), it moves to state S_{34} . At S_{33} , the only licit action for the host node is to process the packet and the engine returns to the initial state S_0 . If the received packet is dropped or forwarded, the engine identifies a malicious behavior and moves to the final state S_{35} . In case that the host node is an intermediate destination (S_{34}), the engine examines whether there is an active route to the final destination, and if such a route exists, it moves to state S_{36} ; otherwise (i.e., if not), it moves to state S_{37} .

At state S_{36} , the expected (i.e., normal) behavior of the host node is to update the routing table entry, while the engine validates the respective modifications: the *lifetime field* of the source, destination, and the next hops (i.e., in both directions) is set up equal to *current time + active route timeout*. After this, the engine moves to state S_{38} . If the host node forwards the data packet, then the engine moves to the state S_{11} (see sect. 3.2.2); otherwise, if the host node drops the packet or the update of routing table deviates from the above, the engine moves to the final state S_{39} . At state S_{37} (i.e., unknown or inactive route to destination), the host node should initiate the local repair mechanism to find an active route by generating a RREQ message, and the engine moves to state S_{10} (see section 3.2.2). If the local repair is successful, the detection engine moves to state S_{36} . In case that the local repair fails, the engine moves to state S_6 and monitors the generation of a RERR message (see Figure 5). Finally, if the host node does not initiate the local repair, the detection engine moves to the final state S_{40} , designating malicious behavior.

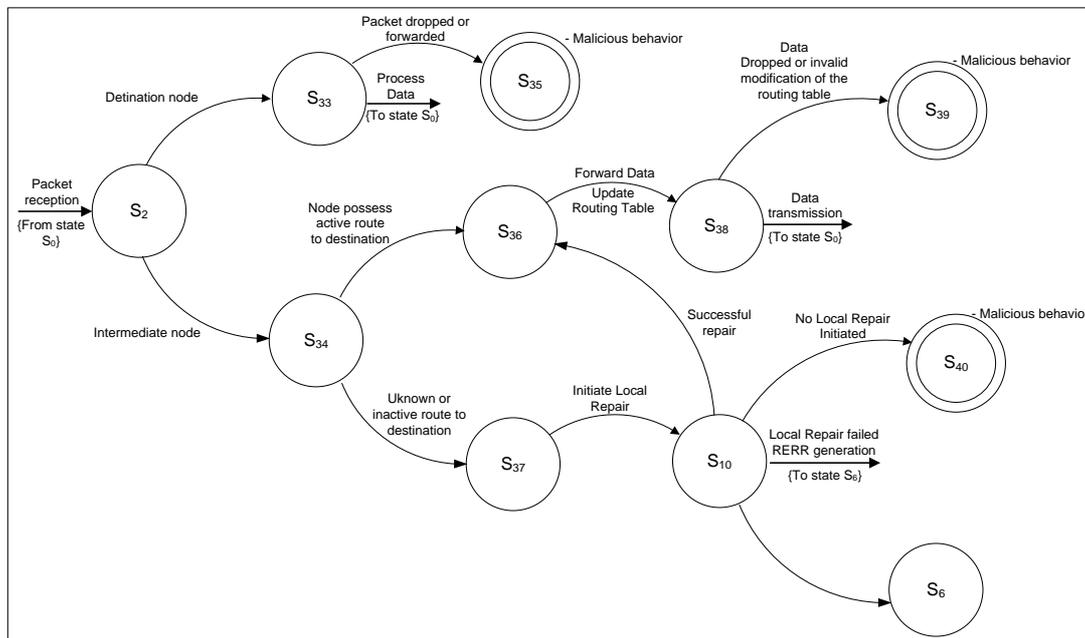


Figure 7: Reception of a data packet

In case that a control message is received (see Figure 8), the engine moves to state S_3 . If the received message is a RERR, the engine moves to state S_6 , where the AODV protocol updates the affected entries of the host node's routing table and provides a list of the

unreachable destinations. This list consists of the unreachable neighbors as well as the destination nodes in the local routing table of the host that use them as next hop. After these, the engine validates the modifications to the routing table's fields, allowing only the following: (i) for every unreachable destination, the *destination sequence number* of its routing entry is set equal to the value of the received RERR message; (ii) the *route entry* is marked as invalid; and (iii) the *lifetime field* is updated to current time plus the *delete period*.

Otherwise (i.e., not a RERR), in S_3 the engine, first, checks that a new routing table entry is created, only, if there is no corresponding entry for the node generating the message. If the received message is a Hello, the routing table entry is always updated, while if it is a RREQ or RREP, the routing table entry is updated, only, under the condition that the received *sequence number* is either higher than the *destination sequence number* in the routing table entry, or are equal, but the received *hop count* incremented by one is smaller than the existing in the routing table. If an update to the routing table takes place (based on the conditions above), the engine validates the modifications to the routing table's fields, allowing only the following: (i) the *destination sequence number* is set equal to the value of the received message; (ii) the *route entry* is marked as valid; (iii) the *lifetime field* in the routing table entry is set equal to the value of the received message (if the message is a RREQ or RREP) or to the value $allowed\ hello\ loss * hello\ interval$ (if the message is a Hello); and iv) the *next hop* field in the respective routing entry depicts the neighbor node from which the message was received. If any deviation from the above takes place, then the engine moves to the final state S_{41} ; otherwise, based on the packet's type it: (i) moves to S_4 (i.e., RREQ message); (ii) moves to S_5 (i.e., RREP message); or (iii) remains at S_3 and starts monitoring the *lifetime timer* (i.e., Hello message). At S_3 , if the host node receives a new Hello message for the considered route entry within the *lifetime interval*, then the engine resets the timer and remains at S_3 ; otherwise, it moves to state S_{42} and marks the route entry as expired.

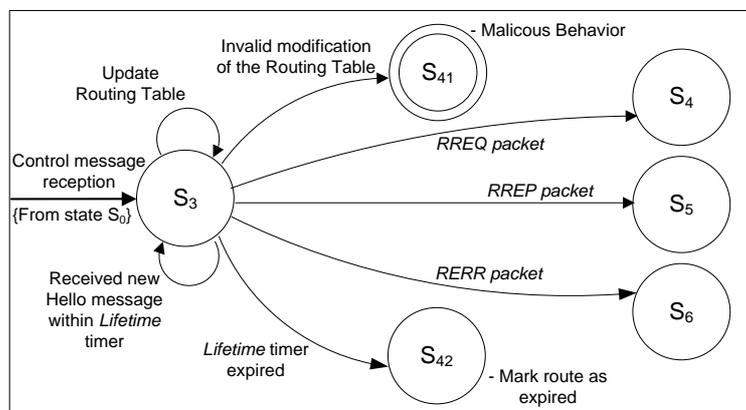


Figure 8: Reception of a control message

Upon a reception of a RREQ (see Figure 9), the engine moves to state S_7 and decides whether the message is to be discarded, based on the following conditions: (i) the neighboring node from which the RREQ was received is blacklisted; or (ii) the host node has already received the same RREQ. If one of the conditions holds, then the engine moves to state S_{43} ; otherwise, it moves to S_{44} . At state S_{43} , the engine checks whether the host node discards the RREQ, and if not, it moves to the final state S_{45} (i.e., malicious behavior). In a similar fashion, at state S_{44} , the engine checks whether the host node retain the RREQ, and then shifts to either state S_{47} (i.e., the host node is the final destination) or S_{48} (i.e., the host node is an intermediate node).

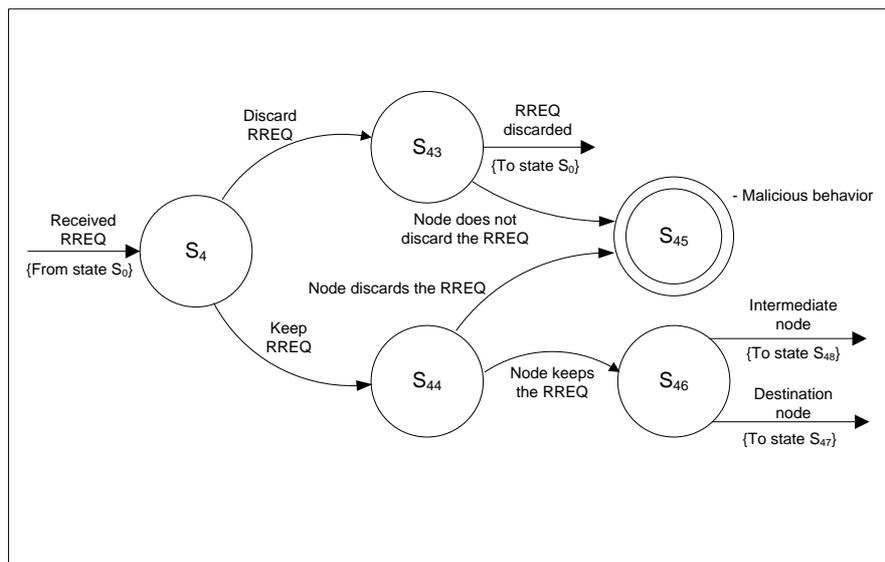


Figure 9: Reception of a RREQ

At state S_{47} (i.e., RREQ reaches the final destination), the engine monitors the generation of a RREP message (see Figure 10) and moves to state S_{49} , where it checks the followings: (i) the host node increments its *sequence number* by one in case that the *sequence number* of the RREQ message is equal to the incremented value, otherwise, the *sequence number* value remains the same; (ii) the *destination IP address* and the *originator sequence number* of the RREP message match to the corresponding fields of the received RREQ message; (iii) the *destination sequence number* field of the RREP message is equal to the host node's *sequence number*; (iv) the *hop count* field is set equal to zero; and (v) the *lifetime* field is set equal to *my route timeout*. At least one invalid value shifts the engine to the final state S_{50} (i.e., malicious behavior), while valid ones to state S_{51} . At S_{51} , the engine monitors whether the host node unicasts the RREP back to the originating node, and if not, it moves to the final state S_{52} . In case that the host node attempts to transmit the RREP, but the neighboring node fails to accept it, the later is added to the host node's blacklist (S_{53}). If not, the detection engine moves to the final state S_{54} .

At state S_{48} (i.e., RREQ reaches an intermediate destination), the engine checks whether the host node has a fresh route to the final destination (see Figure 11) and, if yes, it moves to state S_{55} ; otherwise it moves to S_{56} . At S_{55} , the engine checks if the *destination-only flag* ('D' flag) is set (i.e., means that only the destination node is allowed to generate a RREP), and if so, the engine moves to state S_{56} , monitoring the host node to re-broadcast the RREQ. Any attempt by the host node to generate a RREP will lead to the final state S_{57} . If the 'D' flag is not set, then the detection engine moves to state S_{58} , and, subsequently, checks if the 'G' flag is set (i.e., indicates whether a gratuitous RREP should be sent to the destination node of the RREQ) in the received RREQ packet. If it is, the expected behavior of the host node is to unicast a RREP (i.e., gratuitous RREP) to the RREQ's destination node as well as a RREP message back to the RREQ's originator; otherwise, only the RREP message to the RREQ's originator is expected to be transmitted. If the host node does not transmit the generated RREP message (and the gratuitous RREP if needed), then the engine moves to the final state S_{59} . A successful transmission will move the engine to state S_{60} . Finally, if the host node attempts to transmit the RREP, but the neighboring node fails to accept it, the expected behavior is the inclusion of the neighboring node to the host node's blacklist (S_{61}). If the host node does not include the neighboring node to its blacklist, then the detection engine moves to the final state S_{62} .

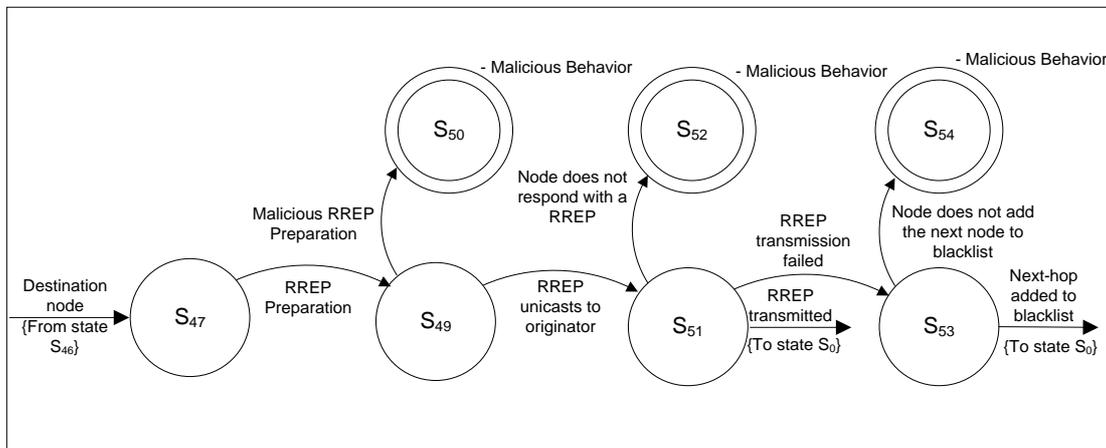


Figure 10: RREQ reaches the final destination

At state S_{56} (i.e., the host node does not possess a fresh route to the destination), AODV modifies the received RREQ message and the engine moves to state S_{63} , allowing only the following changes: (i) the *time to live* is decreased by one, (ii) the *hop count* is incremented by one, and (iii) the *destination sequence number* is set up with the maximum value of the corresponding field of the received RREQ and the one maintained by the node. If the carried modifications are legitimate, the considered message is broadcasted (S_{64}); otherwise the engine moves to the final states: S_{65} (i.e., illegal modifications) or S_{45} (the host node drops the packet).

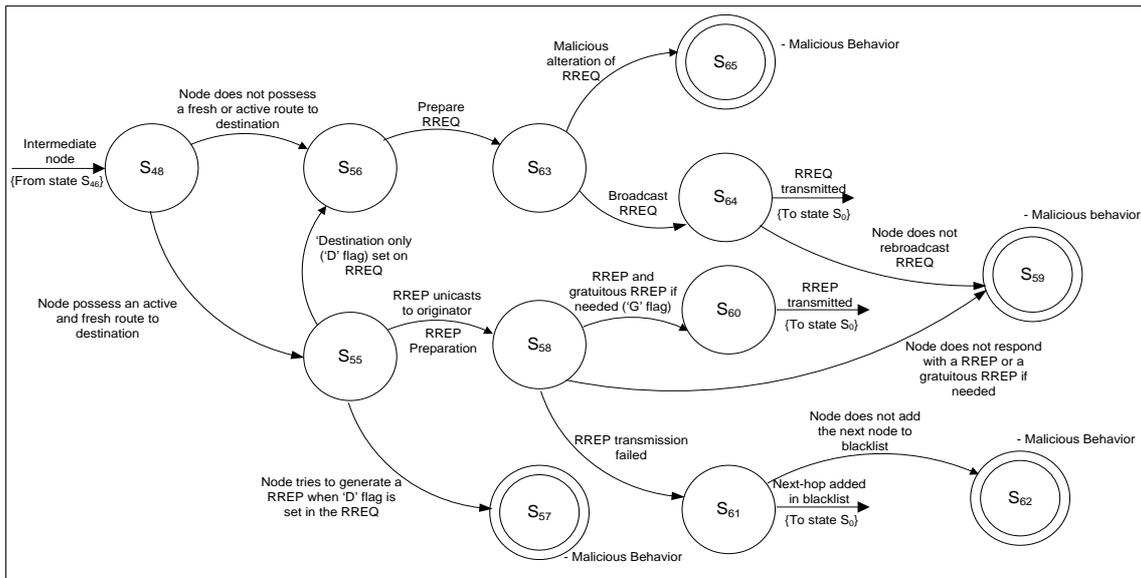


Figure 11: RREQ reaches an intermediate destination

Upon a reception of a RREP message (see Figure 12), the engine moves to state S_5 and shifts either to state: (i) S_9 (i.e., final destination), where the host node is expected to initiate data transfer (see sect. 3.2.2), or (ii) S_{66} (i.e., intermediate), where AODV modifies the received RREP message and the engine moves to state S_{67} . At this state, the engine verifies that the fields of the modified RREP message match to these of the received message except for the *hop count* field, which should be incremented by one. If any of the above is not true, then the engine moves to the final state S_{68} ; otherwise, it moves to state S_{69} , monitoring whether the host node, actually, forwards the RREP towards the destination. In cases that the RREP is dropped, the engine moves to the final state S_{70} , otherwise, the RREP transmission is complete and the engine returns to the initial state S_0 .

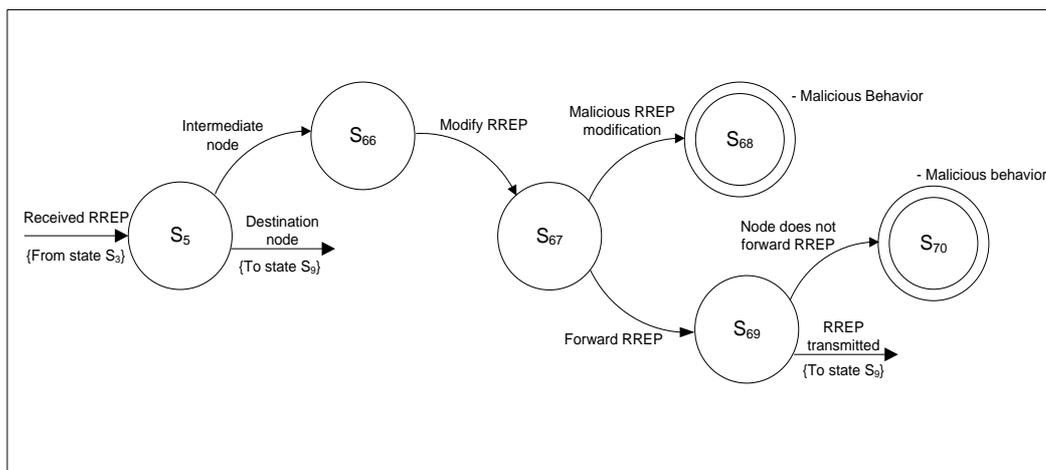


Figure 12: Reception of a RREP

3.3 Remote attestation

The remote attestation procedure enables a node to verify that a particular neighboring node operates an un-tampered version of SIDE. As a prerequisite, each node in the network should store: (i) a 2048bit RSA key pair (N_{PrK} , N_{PuK}); (ii) the network administrator's 2048bit RSA public key (i.e., NA_{PuK}); (iii) an SHA256 hash of NA_{PuK} ; (iv) its own digital certificate $Cert_n$ signed by the network administrator; and (v) SIDE's executable (i.e., the binary code of the detection engine and the remote attestation procedure) in its storage. The "network administrator" may represent a variety of entities ranging from the hardware manufacturer to the network's institutor, depending on the network's deployment objectives, which are out of the scope of this paper. For the sake of simplicity, we assume that all nodes run the same version of SIDE. The node initiating a remote attestation procedure is referred as the "attester" (denoted as N1), while the recipient of the attestation request is the "target node" (denoted as N2). The proposed attestation procedure (illustrated in Figure 13) includes the following steps:

1. Within node N1, SIDE generates a random waiting time (from 0 to $2 \times \text{hello interval}$), before issuing an attestation request to a neighboring node. Subsequently, SIDE keeps track when the random waiting time has elapsed. The interval between attestation requests is based on the *hello interval* parameter because the latter is AODV's optimal parameter for 1-hop control message exchange.
2. Once the waiting timer expires, SIDE randomly selects one of the attester node's neighbors as the target (i.e., node N2) and generates a 128bit random nonce (RN) (i.e., utilized to alleviate replay attacks). It then transmits an attestation request message (i.e., message 1) to the target, which contains the attester's digital certificate $Cert_{N1}$ and the non-predictable RN , as well as generates a SHA256 hash of SIDE's executable residing in N1's storage concatenated with RN (see eq. (1) below).

Message 1: $Cert_{N1}$, RN

3. Upon the reception of the attestation request, node N2 first verifies the authenticity of N1's public key $N1_{PuK}$, using the accompanied certificate $Cert_{N1}$ and the network administrators public key NA_{PuK} . If the public key is authentic, it generates a SHA256 hash of SIDE's executable residing in N2's storage concatenated with RN (see eq. (1) below), and a 128bit AES session key K_{N1N2} .

$$Hash_{SIDE} = SHA256 \{RN || Binary_{SIDE}\} \quad (1)$$

4. The target node N2 then generates an attestation reply message (i.e., message 2), which includes: (i) the target's certificate $Cert_{N2}$; (ii) the generated session key K_{N1N2} as well as

the received RN encrypted, first, by the target's private key $N2_{PrK}$ and, subsequently, by the attester's public key $N1_{PuK}$; and (iii) finally, the hash of SIDE's executable and RN (i.e., $Hash_{SIDE}$) encrypted by the session key K_{N1N2} .

Message 2: $Cert_{N2}, N1_{PuK}\{N2_{PrK}(K_{N1N2}, RN)\}, K_{N1N2}\{Hash_{SIDE}\}$

5. Upon the reception of the attestation reply, the attester node N1, first, verifies the authenticity of N2's public key $N2_{PuK}$, using the accompanied certificate $Cert_{N2}$ and the network administrators public key NA_{PuK} . If it is authentic, it uses its private key $N1_{PrK}$ and N2's public key $N2_{PuK}$ to decrypt the session key K_{N1N2} and RN . It then compares the received RN with the previously generated one, in order to verify that the reply message is legitimate and not a replay attack. Finally, it uses the session key K_{N1N2} to decrypt SIDE's digest $Hash_{SIDE}$ and compares it with its own generated hash. If the target node N2 runs an un-tampered version of SIDE, then the two hashes should match, considering that: (i) all nodes run the same version of SIDE, (ii) SIDE's executable is compiled for the same OS (i.e., the open virtualization kernel), and (iii) both nodes used the same RN value in the computation of (1). If the two hashes do not match, then the target node is considered untrustworthy.

Since nodes N1 and N2 have already established a shared secret session key K_{N1N2} (i.e., known only to nodes N1 and N2), any subsequent attestation requests between them can be performed using this key. The attester node N1 may submit an attestation request to the target node N2, by generating and transmitting RN , while N2's reply encompasses the hash of SIDE's executable concatenated with RN and encrypted with the session key (i.e., $K_{N1N2}\{Hash_{SIDE}, RN\}$). Finally, the session key K_{N1N2} is utilized for as long as nodes N1 and N2 remain neighbors (i.e., while the AODV *lifetime* timer does not expire).

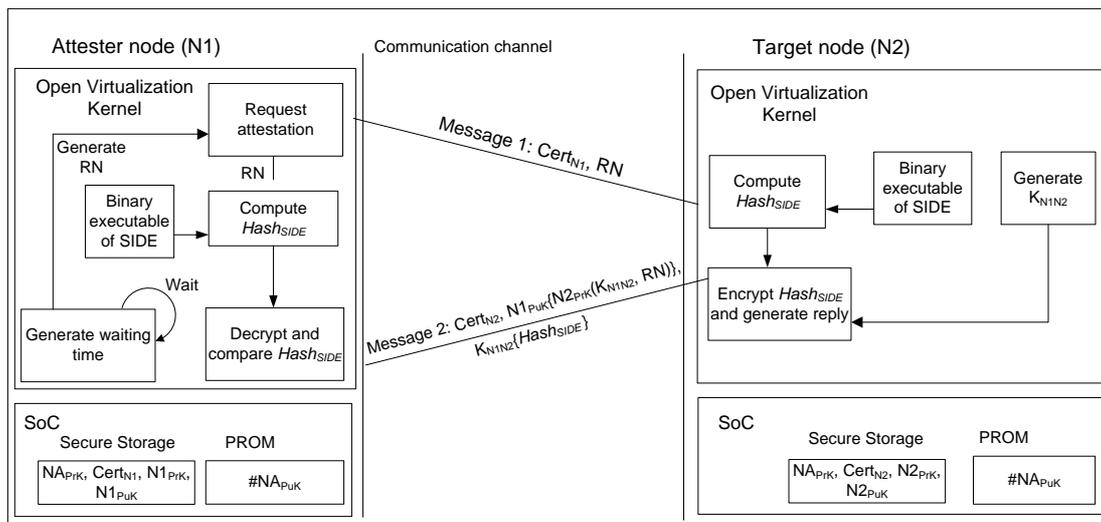


Figure 13: Remote attestation procedure

4 Evaluation of the proposed detection mechanism

In this section, we thoroughly evaluate SIDE focusing on: (i) the advantages over previously proposed mechanisms (see section 4.1); (ii) the robustness of the proposed mechanism against attacks (see section 4.2); (iii) the computational cost and memory requirements (see section 4.3); and (iv) the performance of SIDE compared to other state-of-the-art solutions (see section 4.4).

4.1 Advantages over previously proposed detection mechanisms

SIDE introduces a number of significant advantages over existing detection mechanisms designed for AODV. First off, the employment of the remote attestation procedure enables the engine's deployment in host nodes that may also include suspicious/malicious software, allowing monitoring local information and thus ascertain an accurate view of the protocol operations. Every protocol action is monitored by the employed engine and any deviation or malicious behavior from the protocol specifications is detected, in real-time, minimizing the time in which a malicious activity may induce damage into the network. On the other hand, anomaly-based detection engines, typically, resolve attacks in non-real time, since they have to collect audit data for some predefined time frame, preprocess them, run the detection algorithm, and then, resolve if a malicious activity took place [23]. Therefore, the detection of an attack takes at least:

$$\text{detection time} = TF + P + D \quad (2),$$

where TF is the time frame for collecting audit data, P the preprocessing time, and D the time it takes for the engine to analyze the audit data and provide a decision.

SIDE may also effectively detect every possible attack (i.e., currently known or unknown) that targets the operation of AODV, as long as the attack is expressed as a violation of the protocol's specifications. This is achieved by relying on operational rules developed following the legitimate protocol operation, instructed by the protocol's specifications, rather than attack patterns (i.e., signature-based detection) or statistical behavioral models (i.e., anomaly-based detection). These operational rules, accurately, express the expected protocol behavior and thus, any activity that does not act in accordance with these rules is marked as malicious. Another advantage of SIDE has to do with the fact that its detection accuracy is not negatively affected by network volatility (i.e., churn, changes in the topology, high node's mobility, etc.). This is because the proposed engine does not use the notion of a normal profile, but it monitors the network conditions in real time. On the other hand, in anomaly-based engines, dynamic changes of the network, typically, cause high rate of false positive since they consider them as effects of malicious actions [23].

SIDE's reliance on local information alleviates the associated overheads (i.e., capture, store, and process) of audit data collection, since there is no need for audit data exchange

among network nodes or the monitoring of the exchanged packets. Although the proposed attestation procedure engages neighboring nodes to exchange attestation packets, the imposed communication overhead is limited as presented in sect. 4.3. In addition, the imposed overhead is uniformly distributed among all the network nodes and thus, there is no unfair distribution of detection responsibilities. The utilization of hardware-based encryption minimizes the consumption of the related processing and battery resources, during the attestation procedure. Evaluations of hardware-based encryption have shown that a hardware-based implementation of AES with a key size of 128 bits consumes 97% less energy, compared to software-based encryption; while the throughput is increased by 2500% [45][46].

Finally, SIDE offers several advantages compared to the existing specification-based detection engines for AODV. More specifically, it detects all the possible attacks that target the protocol, because: (a) it monitors all the types of AODV messages, including routing control (RREQ, RREP and RERR) and data packets; and (b) the proposed specifications were developed based on the valid protocol operation and thus, any deviation is considered malicious, regardless if it is a known or unknown attack. On the contrary, the existing specification-based engines proposed in [13][15][16] monitor, only, RREQ and RREP, allowing possible attackers to disrupt routing by generating forged RERR packets. Moreover, the specifications included in the engines analyzed in [14][18] are designed to detect, only, a particular set of AODV attacks, and thus, unknown attacks that do not violate these specifications remain undetected.

4.2 Security evaluation

In an implemented scenario, an adversary may either target the AODV protocol or SIDE. The detection engine of SIDE monitors the protocol's functionality and parameters enabling the detection of any attacks that attempt to violate them. On the other hand, the proposed remote attestation procedure as well as the fact that SIDE is deployed on a TrustZone platform guarantee and safeguard its operation. In the following, we provide a case study of known critical attacks that target the AODV protocol and illustrate how these are resolved by SIDE, in comparison with other specification-based schemes. Moreover, we ascertain the robustness of SIDE against actions that aim to hinder or disable it.

4.2.1 Detection of known attacks

A *RREQ flooding* attack is a commonly used attack, which aims at the consumption of network resources. In this attack, a malicious node broadcasts a large amount of forged RREQ messages with random, fake source and destination IP addresses. Legitimate nodes receiving these messages are obliged to generate reply messages, depleting both their

resources as well as the resources of any other node participating in the route discovery process. SIDE monitors the generation of RREQ messages at each network node (see section 3.2.2) and is capable of detecting whether a host node attempts to perform a RREQ flooding attack. This is achieved by validating the *originator IP address* encapsulated in the generated RREQ and by monitoring if the rate of RREQs per second exceeds the *RREQ rate limit*. RREQ flooding can also be detected by the existing specification-based engines analyzed in [13][14][15][16][18].

Denial of Service (DoS) is another, widely, performed attack in which an adversary attempts to disrupt the network operation and obstruct the access of a legitimate node or set of nodes to the network services. In its simplest form, the malicious node may choose to selectively drop control messages and data packets, aiming at either disrupting specific routes or conserving energy (i.e., selfish node). SIDE, at each host node, keeps track of all packets, from their reception up until transmission, and thus, detects any attempt by the host node to drop, delay, or modify a data or control packet. The attacker may also attempt to break the existing routes, by transmitting forged RERR messages. This will activate the route discovery process, leading to a DoS. Since AODV specifies the circumstances under which a node generates a RERR message, SIDE monitors every message generation, detecting any unjustified attempt (see section 3.2). Existing solutions introduced in [14] and [18] are also capable of detecting such attacks; while the detection engines analyzed in [13][15][16] focus, only, on RREQ and RREP control messages, skipping RRER.

A DoS attack may become more effective in case that a malicious node advertises itself as intermediate to the shortest paths to destinations (i.e., *blackhole attack*). To achieve this, it forwards fabricated RREP messages, which include high sequence numbers, as responds to received RREQs. Once the malicious node starts receiving data packets from the target node(s), it usually drops them. SIDE is capable of detecting if the host node is attempting a blackhole attack, since it monitors: (i) the presence of a fresh route to the requested destination in the routing table; (ii) the eligibility of the host node for generating a RREP; and (iii) the validity of the fields of a generated RREP (including the host's sequence number) (see section 3.3). The engines introduced in [13][14][16][18] are capable of detecting blackhole attack; while this of [15] resolves the attack, only, if the monitoring node receives the legitimate RREP message, issued by the destination. Two colluding malicious nodes may also attract network traffic, by advertising that there is a direct link between them, regardless of the actual distance in hops (i.e., *wormhole attack*). This attack is accomplished by modifying the *hop count* field of the forwarded RREP messages. SIDE detects wormhole attacks, since it monitors all fields (i.e., including the *hop count field*) set during the

generation or modification of a RREP (see section 3.3). This attack is also detected by the engines analyzed in [13][14][15][16], but not from this of [18].

Attackers may attempt to exploit the route discovery process, by omitting the backoff mechanism, performing a *rushing attack*. By forwarding a RREQ packet ahead of time, a malicious node may gain an advantage in being selected as part of an active route. SIDE encompasses a set of rules that specify the valid operation of the backoff mechanism (see section 3.2), and any attempt by the host node to violate these rules will be detected. For example, if the host node attempts to transmit a RREQ before the backoff timer (i.e., $2 * \text{Previous Net Traversal Time}$) expires, it will be flagged as malicious. The detection engines of [14][16] are also capable of detecting this attack; while the engines of [13][15][18] are not.

4.2.2 Robustness of the proposed engine

Adversaries trying to avoid detection may attempt to target SIDE, aiming at hindering its operation, disabling it or tampering its functionality. To carry out these, they may either intervene to the communication channel among networks nodes or have access to the hardware or software of one or more nodes. Intervening in the communication channel, an adversary may masquerade as a legitimate node, by attempting to provide a valid attestation response, replay a previously captured attestation response, or capture, modify and resend one. However, all of these attacks are overcome by the proposed remote attestation procedure. In particular, an adversary cannot masquerade to be a legitimate node, since only the later possesses a valid digital certificate signed by the network administrator. If the adversary attempts to use a self-generated digital certificate, the remote attestation procedure will fail at step 5 (see section 3.3), when the attester nodes tries to authenticate the adversary's digital certificate. Furthermore, the use of a nonce (i.e., the random number RN generated by the attester node) value prevents an adversary from performing a replay attack, since the attester node will reject any response with an outdated nonce value. Finally, the adversary cannot capture a previously transmitted attestation response and modify it in order to forge a reply with the new nonce, since the attestation responses are always encrypted by the target node and the nonce is concatenated within SIDE's hash value.

Having access to a host node, an adversary may also perform software or hardware attacks. Software attacks, typically, aim at altering the behavior of a running program or crashing it, by modifying the execution flow and allowing arbitrary code execution. Such attacks include buffer-overflow, heap overflow, stack smashing, etc. These attacks are counteracted by SIDE using the software isolation mechanisms, employed in the TrustZone SoC. More specifically, the MMU provides a particular interface to the resided OSs (i.e., trusted and un-trusted), enabling each of them to maintain a local table of virtual-to-physical memory address translation. Each entry of these tables includes an NS flag, which is used by

the TrustZone CPU to identify if the equivalent memory address belongs to either the trusted or the un-trusted OS and prevent the later from accessing any memory area allocated to the trusted OS.

Hardware attacks attempt to exploit platform vulnerabilities, which are related to a hardware interface commonly used for testing and debugging, known as joint test action group (JTAG). JTAG is embedded in most processors, devised for testing and debugging circuit boards. It may provide access to an adversary to resources that are only accessible by the trusted OS (i.e., memory, storage, co-processor registers, etc). However, a TrustZone SoC comes with the provision of permanent deactivation or semi-deactivation of this interface, during manufacturing. A permanent deactivation completely disables it; while a semi-deactivation limits its access to resources that are available to the un-trusted OS.

4.3 Computational cost and memory requirements

To quantify the computational cost and memory requirements of SIDE, both the remote attestation procedure and the detection engine should be individually analyzed. The remote attestation procedure does not induce any significant computational costs, since all of the required cryptographic computations are performed by the TrustZone crypto-coprocessor [42],[50]. On the other hand, the remote attestation procedure induces memory consumption, due to the fact that additional memory sections are allocated to store the required cryptographic parameters. That is, 1 KB for the Certificate, 512 bytes for the RSA key pair, 256 bytes for the network administrator's public key, 16 bytes for RN, 16 bytes for the session key, and 32 KB for the hash digest (see section 3.3). In total, 1856 bytes are allocated and stored in the memory for the remote attestation procedure.

The detection engine induces both computational costs and memory consumption when it utilizes the CPU and the memory to perform comparisons, in order to check if the specifications are violated or not. To quantify the computational costs, first we determine the maximum number of comparisons performed by the detection engine, while executing a complete set of specifications. As mentioned in section 3.2, the specifications are divided into three sets, based on the host-node's communication condition: a) idle, b) transmitting, and c) receiving. Next, we calculate how many CPU instructions are required to perform these comparisons for each one of the above three sets of specifications.

Let U_I , U_T , and U_R denote the maximum number of comparisons made by the detection engine for the: a) idle, b) transmitting, and c) receiving sets of specifications, respectively. The number of CPU instructions required for a comparison depends on the target CPU architecture. In this analysis, we assume that the ARM CPU implements the thumb-2 instruction set [51], which requires one CPU instruction to perform a single comparison between two integer values, stored in the CPU's registers [51]. Three additional CPU

instructions are required to fetch the two integers into the registers and store the result back into the memory. Thus, each comparison is estimated at a cost of four CPU instructions. Based on the above, the CPU instructions required for the execution of the maximum number of comparisons for each set of specification is given by:

$$C_I = 4U_I \quad (4),$$

$$C_T = 4U_T \quad (5),$$

$$C_R = 4U_R \quad (6),$$

where C_I , C_T , C_R denote the maximum number of CPU instructions for the idle, transmitting, and receiving sets of specifications, respectively.

In order to calculate the values of U_I , U_T , and U_R , we measure the highest number of comparisons performed by each respective set of specifications, when all their states are executed. The maximum value of U_I is 3, since the idle specification verifies three conditions (see section 3.2.1): (i) if an incoming data packet is ready for transmission, (ii) if a data packet has been received, and, (iii) if a control packet has been received. On the other hand, U_T has a maximum value of 28 comparisons, which occurs when a data packet is ready for transmission, but there is no route to the destination and the route repair process fails (see section 3.2.2). Finally, the maximum value of U_R is 39 comparisons and it is reached when a RREQ message is received by an intermediate destination and the 'G' flag on the RREQ message is set, forcing the generation of a gratuitous RREP (see section 3.2.3). Based on these measurements, we determine that the maximum number of CPU instructions required for the three sets of specifications are: $C_I = 12$, $C_T = 112$, and, $C_R = 156$.

On the other hand, to quantify the detection engine's memory consumption, we have calculated the memory required for storing the maximum number of parameters that may be used by the detection engine, during the execution of a set of specifications (i.e., idle, transmitting, or receiving). We assume these parameters are stored as 32-bit integers. For each parameter, there are two values that are stored in memory: one value observed by the engine while monitoring the AODV protocol and one value which is the expected (i.e., legitimate) value, based on the specifications. In any AODV operation, the detection engine may allocate memory to temporarily store the following parameters: the destination IP address, the originator IP address, the destination sequence number, the originator sequence number, the RREQ id, the hop count field, the RREQ rate limit, the net traversal time, the RREQ retries counter, the TTL max counter, the route entry flag, the lifetime field, the no delete, 'G', 'D' flags, the RERR rate limit, the active route timeout, the blacklist timeout, the allowed hello loss and the hello interval. Based on the above, the maximum number of parameters allocated in the memory during the execution of the detection engine is 40, resulting in a total memory consumption of 160 bytes.

4.4 Performance Evaluation

In this section, a prototypical implementation of SIDE is evaluated through simulations. The objective of the carried simulations was to compare SIDE's performance to other security mechanisms designed for AODV, verifying the previously mentioned advantages.

The simulations were performed using the ns-3 network simulator [35], version 3.16, which was tested and validated after the installation. The underlying network topology was constructed by, randomly, placing 25, 50, 75, and 100 nodes in an area of 1000m x 1000m, where nodes established links if they were in a radio range of 100m or less. Network traffic was generated by 10 randomly selected constant bit rate (CBR) source nodes, which transmitted 512 bit data packets at fixed rates of 5, 10, or 20 packets per second. Nodes' mobility was simulated using the random waypoint mobility model and the speed of mobile nodes ranged from 0 to 20 m/s. Table 2 displays a summary of the simulation parameters."

Table 2: Simulation parameters

Simulation parameters	Value
Number of nodes	25/50/75/100
Simulation area	1000 m x 1000 m
Radio range	100 m
Mobility model	Random waypoint
Nodes' mobility	Ranged from 0 to 20 m/s
Channel capacity	2Mbps
Traffic type	CBR
Traffic volume	5/10/20 packets per second
CBR packet size	512 bytes

The comparative evaluation, except from SIDE, includes results from: (i) the SAODV protocol [36]; (ii) three anomaly-based intrusion detection engines, SVM [26], MRF1 [30], and DCM [37]; as well as (iii) the pure AODV without any security mechanism. The latter is used as a base, where the imposed overheads by the considered security mechanisms can be compared and studied. SAODV was selected because it constitutes a solution that encompasses security features within AODV. Moreover, the three anomaly-based detection engines were selected since they utilize state-of-the-art techniques (i.e., thresholds and dynamic profiling) as well as they provide a broad set of simulation results, allowing for a detailed comparison with SIDE.

To quantify the performance and facilitate the comparison among the studied solutions, we have selected the following metrics: (i) the packet deliver ratio (i.e., the percentage of transmitted packets that reach their destination); (ii) the imposed control packet overhead (i.e., the increase in the percentage of control packets transmitted by AODV); and (iii) the observed detection accuracy (i.e., the percentage of attacks that are resolved by the detection mechanism). The packet delivery ratio, in the presence of a malicious node(s) performing a packet dropping attack(s), may assess the ability of the considered mechanism to detect and isolate malicious nodes, minimizing the effects of the attack(s) in the operation of the network. The control packet overhead measures the corresponding control packets initiated by the considered mechanism, determining the induced communication, processing and energy consumption overhead. Finally, the detection accuracy of a mechanism, in the presence of a sinking attack(s) and under variable nodes' mobility, evaluates the mechanism's ability to resolve attacks in volatile network conditions.

Figure 14 presents the packet delivery ratio as a function of the number of malicious nodes, performing a packet dropping attack for AODV, SIDE, SAODV, and DCM. SVM and MRF1 are excluded from this study, since their authors do not provide such results for their performance. In the presence of one malicious node, the packet delivery ratio of pure AODV (i.e., without any security measure) is about 80%; while for 10 malicious node the ratio drops down to 20%, depicting the impact of the attack. The packet delivery ratio for SIDE is approximately 100%, regardless of the number of malicious nodes, the density of network nodes, or the volume of generated traffic. These results confirm the assertion that SIDE performs detection in real time, meaning that attacks are detected immediately after they are initiated by a malicious node. This can be attributed to the fact that SIDE monitors the behavior of its host node, relying exclusively, on local information. Thus, any change in the number of malicious nodes, the volume of traffic or the density of nodes, does not have an impact on the monitoring process and the detection capabilities of SIDE. SAODV demonstrates similar performance with SIDE, which in the worst case scenario of 10 malicious nodes it presents a packet delivery rate of 95%. This occurs because SAODV eliminates the ability of malicious nodes to generate false RREP, allowing only the final destinations of a RREQ to do this. Nevertheless, the malicious nodes are still capable of dropping some data packets transmitted. On the contrary, the employment of the DCM engine degrades the network operation and the considered metric as the number of malicious nodes increases from 1 to 10, reaching the lowest value of 80% for the presence of 10 malicious nodes.

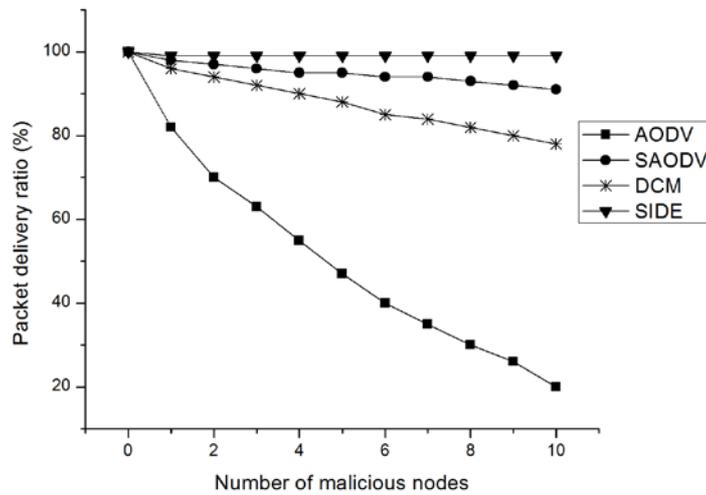


Figure 14: Packet delivery ratio as a function of the number of malicious nodes

Figure 15 depicts the percentage of additional control packets, compared to pure AODV, transmitted by SIDE, SAODV, and DCM, as a function of the total number of transmitted packets. Once again, SVM and MRF1 are excluded from this study, since their authors do not provide comprehensive results for their performance. SIDE induces the least control packet overhead (averaging 6%), and its performance is close to that of the AODV protocol running without any security mechanisms. Moreover, we observe that the control packet overhead of SIDE remained constant when the nodes density is increased. Even though the higher number of nodes results in additional SIDE's control packets (i.e., additional attestation messages), at the same time it causes a similar increase in control messages for AODV. Thus, the relative control packet overhead between the proposed mechanism and AODV is not modified when the nodes density increases. Moreover, the employment of SAODV in a network of 50 nodes results in a control message overhead of about 20%. On the other hand, as shown in figure 15, DCM exhibits the highest amount of control packet overhead, exceeding on average 100% in a network of 50 nodes and 150% in a network of 100 nodes. However, it is the only scheme that diminishes the occurred overhead as the total number of transmitted packets increases. This is mainly caused by the mechanisms' ability to provide more accurate decisions about the behavior of nodes as the respective number of monitoring packets increases. In this way, the need to transmit "test packets" between monitoring nodes are minimized, reducing the associated communication overheads.

We have also estimated the control packet overhead of SIDE when the traffic volume is increased. Numerical results showed that this overhead decreases when the volume of traffic is increased. In particular, a traffic volume of 5, 10, and 20 packets per second results in a

packet control overhead of 11%, 6.1%, and 2.8%, respectively. This happens because the control packets of SIDE (i.e., remote attestation messages) are generated at fixed interval times, independently of the traffic volume. On the other hand, the increase of traffic volume results in additional AODV control messages. As a result, the SIDE's control packet overhead in proportion to the number of AODV control messages is decreased.

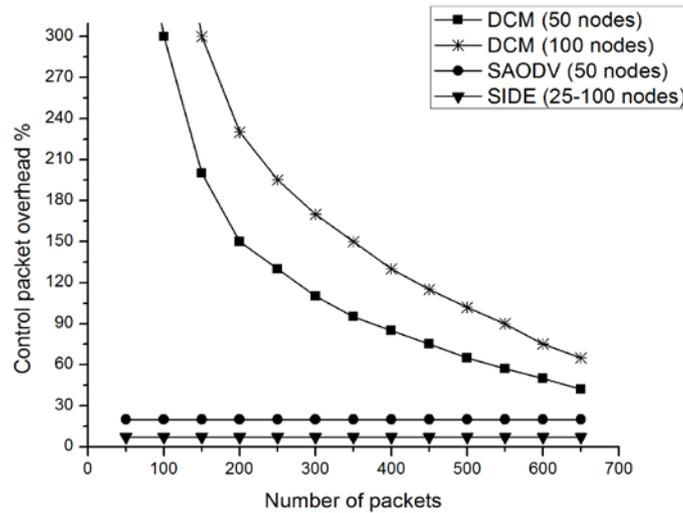


Figure 15: Control packet overhead as a function of the number of packets

It is worth noting that SIDE's control packets are smaller in size compared to AODV and SAODV. More specifically, during the remote attestation procedure of SIDE, two packets are exchanged with a size of approximately 1040 bytes (i.e., 1KB for the Certificate and 16 bytes for RN) and 1536 bytes (i.e., 1KB for the certificate, 256 bytes for the encrypted session key, and 256 bytes for the encrypted hash), respectively. In total, around 2576 bytes are transmitted per attestation request, or roughly around 1/10 of the size (i.e., 25 KB) of an AODV control packet. The size of SAODV control packets is almost twelve (12) times the size of the original AODV control packets (i.e., around 300 Kb in comparison to 25 KB for pure AODV). This is due to the additional fields required by SAODV for the conveyance of digital signatures and hashes. Finally, the authors of DCM do not provide any information related to the size of the "test packets", transmitted by the monitoring nodes.

Figure 16, presents the detection accuracy (i.e., the percentage of attacks that are resolved by the detection mechanism) of SIDE and DCM, as a function of the number of malicious nodes performing a blackhole attack. SVM, MRF1, and SAODV are excluded from this study, since their authors do not provide respective results for their performance. Overall, we conducted ten iterations of this experiment, in each of which the number of randomly selected malicious nodes was incremented by one. The employment of SIDE results in a detection accuracy of 100%, regardless of the amount of malicious nodes in the network, the

density of nodes, or the volume of traffic. This can be attributed to the fact that SIDE relies on a host-based architecture and thus, the presence of a higher number of malicious nodes does not affect its detection accuracy. On the other hand, the deployment of DCM in a network with nodes density equal to 100, results in a slight drop of the detection accuracy, which reaches 99% in the presence of 10 malicious nodes. When the nodes density is decreased to 50, then the detection accuracy of DCM drops even further, and reaches 91% (i.e., the number of malicious nodes is 10). This happens because decreasing the nodes density, limits the number of cooperating nodes (i.e., legitimate), thus hindering the ability of DCM to resolve malicious activities.

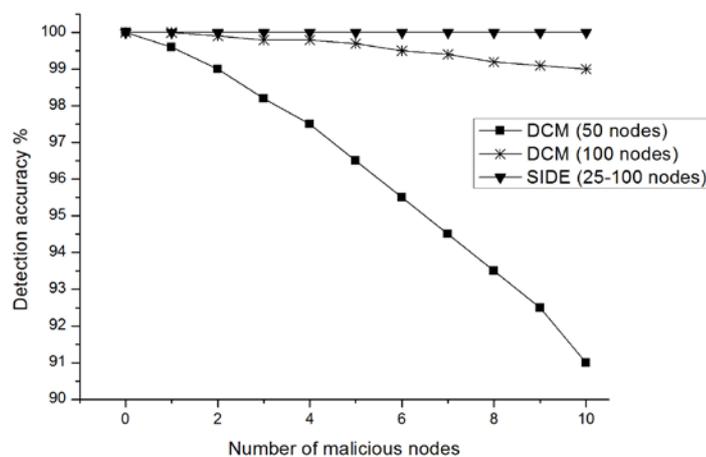


Figure 16: Detection accuracy as a function of the number of malicious nodes

Figure 17 presents the detection accuracy of SIDE, SVM, and MRF1 as a function of the average node's speed. SAODV and DCM are excluded from this study, since their authors do not provide such results for their performance. We conducted ten iterations of this experiment, in each of which we increased the average node speed by a factor of 2 m/s, beginning with an initial value of 0 m/s. For each iteration, ten nodes were randomly selected to perform a sinking attack. The employment of SIDE results in a detection accuracy that is constant and equal to 100%, regardless of the volatile network conditions (i.e., nodes movement, density of nodes, or volume of traffic). This can be attributed to the fact that SIDE resides at each network node, monitoring the nodes behavior and action using a well-defined set of rules, which represent the legitimate behavior and actions, based on the AODV specifications. The deployment of SVM also leads to comparable results with average detection accuracy around 96%. This is because the engine uses a dynamically updated normal profile, which however, introduces a number of limitations (see section 2.2). On the other hand, the detection accuracy of MRF1 is highly affected by the change in node's mobility and with an average node's speed of 15 m/s, it drops more than 20%. MRF1 utilizes

a static normal profile and hence it is more prone to be affected by dynamic network changes. These changes may cause MRF1 to rely on outdated information (i.e., normal profile) and thus perform poorly.

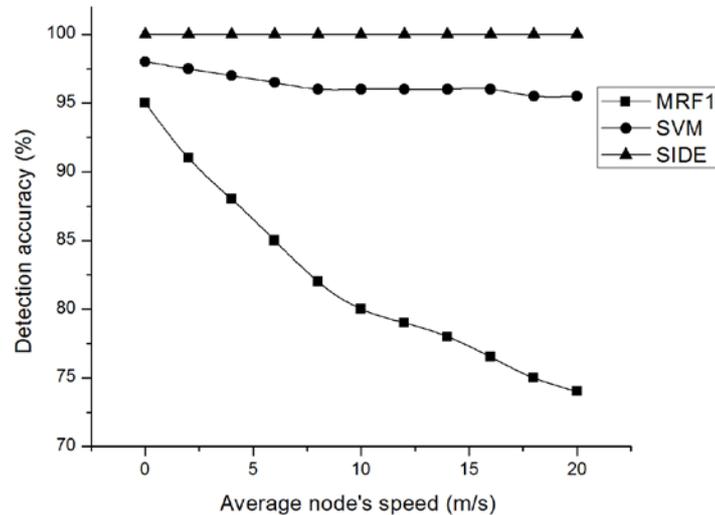


Figure 17: Detection accuracy as a function of the average node's speed

5 Conclusions

This paper proposed a monitoring mechanism called SIDE that safeguards the operation of AODV. SIDE encompasses two complementary functionalities: (i) a specification-based detection engine for AODV, and (ii) a remote attestation procedure. It operates on a trusted computing platform, which provides hardware-based root of trust and cryptographic acceleration used by the remote attestation procedure, as well as protection against runtime attacks. A key advantage of the proposed mechanism is its ability to effectively detect both known and unknown attacks. The performance analysis showed that SIDE is capable of detecting malicious behaviors in real time, thus minimizing the impact of an attack. It was also observed that it resolves attacks with a low percentage of false positives/negatives even under high network volatility. Moreover, SIDE induces the least amount of control packet overhead in comparison with a number of other proposed IDS schemes.

References

- [1] WiFi Alliance, "Wi-Fi Peer-to-Peer (P2P) Technical 7 Specification," www.wi-fi.org/Wi-Fi_Direct.php
- [2] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," IETF RFC 3561, Jul. 2003.
- [3] D. Cavalcanti, D. Agrawal, C. Corderio, B. Xie, and A. Kumar, "Issues in integrating cellular networks WLANs, and MANETs: A futuristic heterogeneous wireless network," *IEEE Wireless Communications*, vol. 12, no. 3, pp. 30–41, Jun. 2005.

- [4] J. Ott, D. Kutscher, and C. Dwertmann. Integrating DTN and MANET Routing. In CHANTS '06: in proceedings of the 2006 SIGCOMM Workshop on Challenged Networks, pages 221–228, New York, NY, USA, 2006.
- [5] J. Lakkakorpi, M. Pitk"anen, and J. Ott, "Adaptive routing in mobile opportunistic networks," in proceedings of the ACM MSWIM'10, (Bodrum, Turkey), pp. 101–109, October 2010.
- [6] A.A. Pirzada, M. Portmann, J. Idulska, "Evaluation of Multi-Radio Extensions to AODV for Wireless Mesh Networks", in proceedings of the international workshop on Mobility management and wireless access ACM MobiWac 2006.
- [7] C. Gomez, P. Salvatella, O. Alonso, J. Paradells, "Adapting AODV for IEEE 802.15.4 mesh sensor networks: theoretical discussion and performance evaluation in a real environment," in proceedings of the 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06), 2006, pp. 159–170.
- [8] C. Xenakis, C. Panos, I. Stavrakakis, "A comparative evaluation of intrusion detection architectures for mobile ad hoc networks," *Computers & Security*, Volume 30, Issue 1, January 2011.
- [9] B. Wu, J. Chen, J. Wu, and M. Cardei, "A Survey of Attacks and Countermeasures in Mobile Ad Hoc Networks" in "Wireless Network Security", Y. Xiao, X. Shen, and D. -Z. Du, Springer, Network Theory and Applications, Vol. 17, 2006.
- [10] C., Panos, P., Kotzias, C. Xenakis, I. Stavrakakis, "Securing the 802.11 MAC in MANETs: A Specification-based Intrusion Detection Engine," 9th Annual Conference on Wireless On-demand Network Systems and Services (WONS), 2012
- [11] P. Ning and K. Sun, "How to misuse AODV: a case study of insider attacks against mobile ad-hoc routing protocols," *Ad Hoc Networks*. 3, 6 (November 2005), 795-819.
- [12] T. Anantvalee, J. Wu, "A Survey on Intrusion Detection in Mobile Ad Hoc Networks," *Wireless/Mobile Network Security*, Springer, Chapter 7, pp. 170 - 196, 2006.
- [13] C.-Y. Tseng. et al., "A specification-based intrusion detection system for AODV," in proceedings of ACM Workshop on Security of Ad Hoc and Sensor networks, 2003.
- [14] Y. Huang and W. Lee, "Attack Analysis and Detection for Ad Hoc Routing Protocols," in proceedings of the 7th international symposium in recent advances in intrusion detection (RAID 2004), Sophia Antipolis, France, Sept. 2004.
- [15] H.M. Hassan, M. Mahmoud, and S. El-Kassas, "Securing the AODV Protocol using Specification-based Intrusion Detection," in proceedings of the 2nd ACM International Workshop on Quality of Service & Security for Wireless and Mobile Networks, Torremolinos, Spain, 2006.
- [16] J. Grönkvist, A. Hansson, M. Sköld, "Evaluation of a Specification-Based Intrusion Detection System for AODV," in The Sixth Annual Mediterranean Ad Hoc Networking WorkShop. 2007: Corfu, Greece. p. 121-128.
- [17] V. Mulert, J. I. Welch, and W. KG Seah. "Security threats and solutions in MANETs: A case study using AODV and SAODV," *Journal of Network and Computer Applications* 35.4 (2012): 1249-1259.
- [18] G. Vigna, S. Gwalani, K. Srinivasan, E. Belding-Royer, R. Kemmerer, "An Intrusion Detection Tool for AODV-based Ad Hoc Wireless Networks," in proceedings of the Annual Computer Security Applications Conference (ACSAC), pages 16–27, Tucson, AZ, December 2004.
- [19] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Standard 802.11 - 2007.
- [20] Trusted Computing Platform Alliance. TCPA main specification v. 1.2. <http://www.trustedcomputing.org/>.
- [21] M.Shaneck, K.Mahadevan, V.Kher, and Y.Kim. "Remote software-based attestation for wireless sensors," *Security and Privacy in Ad-hoc and Sensor Networks*, Vol. 3813, pp 27–41, 2005.
- [22] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - tools for software protection," In *IEEE Transactions on Software Engineering*, volume 28, pages 735–746, August 2002.
- [23] C. Panos, C. Xenakis, I. Stavrakakis, "An evaluation of anomaly-based intrusion detection engines for mobile ad hoc networks," in proceedings of the 8th International Conference on Trust Privacy and Security in Digital Business (TrustBus 2011), Toulouse, August 2011.
- [24] R. Kennell and L. H. Jamieson, "Establishing the genuinity of remote computer systems." in the 12th USENIX Security Symposium, pages 295–310. USENIX Association, August 2003.

- [25] H., Nakayama, S., Kurosawa, A., Jamalipour, Y., Nemoto, N., Kato, "A Dynamic Anomaly Detection Scheme for AODV-Based Mobile Ad Hoc Networks," *Vehicular Technology, IEEE Transactions on*, vol.58, no.5, pp.2471-2481, Jun 2009.
- [26] J.F.C., Joseph, Bu-Sung Lee, A., Das, Boon-Chong Seet, "Cross-Layer Detection of Sinking Behavior in Wireless Ad Hoc Networks Using SVM and FDA," *Dependable and Secure Computing, IEEE Transactions on*, vol.8, no.2, pp.233-245, March-April 2011.
- [27] C. Nello and S.-T. John, "An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods," Cambridge Univ. Press, 2000.
- [28] A. Lauf, R. A. Peters, W. H. Robinson, "A Distributed Intrusion Detection System for Resource-Constrained Devices in Ad Hoc Networks". *Elsevier Journal of Ad Hoc Networks*, vol. 8, issue 3, pp. 253-266, May 2010.
- [29] P. Kabiri and M. Aghaei, "Feature Analysis for Intrusion Detection in Mobile Ad-hoc Networks," *International Journal of Network Security*, Vol.12, No.2, PP.80–87, Mar. 2011.
- [30] A., Nadeem, M., Howarth, "Adaptive intrusion detection and prevention of denial of service attacks in MANETs," in *International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly* (pp. 926–930). Leipzig, Germany, 2009.
- [31] C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Principles of Programming Languages 1998, POPL'98*, San Diego, CA, Jan. 1998.
- [32] A. Seshadri et al., "SCUBA: Secure Code Update by Attestation in Sensor Networks," *Proc. 5th ACM Wksp. Wireless Security (WiSe'06)*, Los Angeles, Sept. 2006.
- [33] Federal Information Processing Standards Publication (FIPS PUB) 180-4, Current version of the Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), March 2012.
- [34] T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji. "Software tamper resistance based on the difficulty of interprocedural analysis," *Proc 3rd Workshop on Information Security Applications (WISA 2002)*, Korea, August 2002.
- [35] The NS-3 Network Simulator, <http://www.nsnam.org/>. Last accessed at 17/03/2014.
- [36] M. Zapata and N. Asokan, "Securing Ad-hoc Routing Protocols," in *Proc. of ACM Workshop on Wireless Security (WiSe)*, Atlanta, GA, Sept. 2002.
- [37] C.W. Yu, T.K. Wu, R. H., Cheng, and S. C. Chang, "A Distributed and Cooperative Black Hole Node Detection and Elimination Mechanism for Ad Hoc Networks", *PAKDD 2007 Workshops*, pp. 538–549, 2007.
- [38] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang. "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence." In *Proceedings of the 39th International Conference on Dependable Systems and Networks (DSN'09)*, 2009.
- [39] Seshadri, A., Luk, M., and Perrig, A. "SAKE: Software attestation for key establishment in sensor networks." In *Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems (DCOSS 2008)*.
- [40] L. Davi, A. R. Sadeghi, and M. Winandy. "Dynamic integrity measurement and attestation: Towards defense against return-oriented programming attacks." In *Proceedings ACM workshop on Scalable trusted computing*, 2009.
- [41] J. Clause, W. Li, A., Orso, "Dytan: A Generic Dynamic Taint Analysis Framework. In *Proceedings of the 2007 international symposium on Software testing and analysis*, 2007.
- [42] ARM Ltd. ARM1176JZF-S Technical Reference Manual, Revision: r0p7. Available online at: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301g/DDI0301G_arm1176jzfs_r0p7_trm.pdf, 2008.
- [43] The Open Virtualization kernel, <http://www.openvirtualization.org/>. Last accessed at 08/06/2013.
- [44] TEE Internal API Specification v1.0 <http://www.globalplatform.org/specificationsdevice.asp>. Last accessed at 08/06/2013.
- [45] I. Ekelund, "Low energy AES hardware for microcontroller." PhD diss., Norwegian University of Science and Technology, 2009.
- [46] V. Cervenka, "Energy Efficient Public Key Cryptography in Wireless Sensor Networks." *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering*. Springer New York, 2013. 497-509.
- [47] W. Svante, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems 2.1* (1987): 37-52.
- [48] Johnson, David B., David A. Maltz, and Josh Broch. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks." *Ad hoc networking 5* (2001): 139-172.
- [49] C. Perkins, B. Pravin, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers." *ACM SIGCOMM Computer Communication Review*. Vol. 24. No. 4. ACM, 1994.

- [50] A. Hodjat, V. Ingrid, "Interfacing a high speed crypto accelerator to an embedded CPU," Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on. Vol. 1. IEEE, 2004.
- [51] ARM Architecture Reference Manual, Thumb-2 Supplement. Available online at: <http://read.pudn.com/downloads159/doc/709030/Thumb-2SupplementReferenceManual.pdf>