# A Moving Target Defense Mechanism for MANETs Based on Identity Virtualization

Massimiliano Albanese*, Alessandra De Benedictis†, Sushil Jajodia*, and Kun Sun*
*Center for Secure Information Systems
George Mason University, Fairfax, VA 22030, USA
Email: {malbanes,jajodia,ksun3}@gmu.edu
†Department of Computer Science
University of Naples Federico II, Naples, NA 80125, Italy
Email: alessandra.debenedictis@unina.it

***Abstract*—Mechanisms for continuously changing or shifting a system's attack surface are emerging as game-changers in cyber security. In this paper, we propose a novel defense mechanism for protecting the identity of nodes in Mobile Ad Hoc Networks and defeat the attacker's reconnaissance efforts. The proposed mechanism turns a classical attack mechanism – Sybil – into an effective defense mechanism, with legitimate nodes periodically changing their virtual identity in order to increase the uncertainty for the attacker. To preserve communication among legitimate nodes, we modify the network layer by introducing (i) a translation service for mapping virtual identities to real identities; (ii) a protocol for propagating updates of a node's virtual identity to all legitimate nodes; and (iii) a mechanism for legitimate nodes to securely join the network. We show that the proposed approach is robust to different types of attacks, and also show that the overhead introduced by the update protocol can be controlled by tuning the update frequency.**

## I. Introduction

Network reconnaissance is the first step in cyber-attacks mounted by stealthy, resource-aware and intelligent adversaries. Reconnaissance enables an adversary to gather information about the network topology and dynamics as well as other critical information about the target system. This information can be used to identify system vulnerabilities, and design and execute specific exploits on the system or services. Therefore, thwarting the network reconnaissance step is critical for preventing further attack steps. To this aim, Moving Target Defense (MTD) [1] is emerging as a game-changing approach consisting in a number of mechanisms that automatically change one or more system attributes in order to make a system's attack surface [2] unpredictable to adversaries. As stated by the Executive Office of the President, National Science and Technology Council [3], Moving Target Defense *"enables us to create, analyze, evaluate, and deploy mechanisms and strategies that are diverse and that continually shift and change over time to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency"*.

A well-designed MTD mechanism ensures that, at any given time, an adversary cannot easily discover a specific entry point to the system or specific protocols that could be exploited to compromise it. Ideally, MTD aims at making the random attack strategy the most effective strategy for the attacker.

In this paper, we focus on Mobile Ad Hoc Networks (MANETs), which are attracting considerable interest, especially in military communications, as they offer network capabilities which are readily deployable, self-organizing, robust to failures of individual nodes, and able to adapt to frequent topology changes triggered by node mobility, varying radio conditions, or hostile intervention. Given their wireless nature, MANETs are prone to passive reconnaissance attacks aimed at reconstructing the topology of the network or analyzing traffic flows and mobility patterns. Moreover, information gathered from captured control and routing messages could be leveraged to run a wide range of active attacks. For instance, attackers could forge malicious routing messages aimed at disrupting legitimate communications. For these reasons, it is crucial to hide node identities and routing information from adversaries, but current solutions do not provide similar guarantees.

Based on the above considerations, we propose an MTD approach to improve the security of MANETs by increasing an attacker's uncertainty about the topology of the network. The proposed approach consists in periodically changing the identity that legitimate nodes present to other nodes – referred to as the *virtual identity* – and securely informing them about the change. The proposed mechanism turns a classical attack mechanism – the *Sybil attack* [4] – into an effective defense mechanism. In Sybil, a malicious node can forge and use multiple identities in order to subvert the reputation system of a peer-to-peer network. Similarly, in the proposed defense approach, legitimate nodes periodically change their virtual identity in order to defeat the attacker's reconnaissance efforts. Each node has a unique real identity and a pool of virtual identities. Such pool can be generated in different ways, but, in this paper, we propose the use of hash chains to generate a pool such that future identities are hard to predict for the attacker, while all legitimate nodes can readily map virtual identities of every other node to the corresponding real identity. Legitimate nodes communicate using only virtual identities. To preserve their ability to communicate with one another in spite of frequent identity changes, we modify the network layer by introducing a *translation service* to map virtual identities to real identities, and develop a protocol for propagating updates of a node's virtual identity to all legitimate nodes and an ad-hoc mechanism for legitimate nodes to securely join the network.

The work presented in this paper is supported in part by the Army Research Office under award number W911NF-12-1-0448, and by the Office of Naval Research under awards number N00014-12-1-0461 and N0014-11-1-0471

In this paper, we show that the proposed approach is effective in preventing or mitigating several types of classical external attacks against the routing protocol. Additionally, as the translation service itself – along with the update and join protocols – may become the target of an attack, we demonstrate its resilience to both packet delays or losses due to mobility and specific attacks aimed at compromising its operation.

We implemented the proposed defense mechanism in the ns-2 simulator, and evaluated its performance under several traffic and mobility conditions. As expected, a trade-off exists between the update frequency – which in turns influences the level of security achieved – and the resulting overhead and performance. Our simulation results show that the overhead introduced by the update protocol can be controlled by tuning the update frequency. As the size of the network increases, lower update frequencies should be chosen to maintain the overhead within acceptable limits. However, this is reasonable, as the effort required from an attacker to gain knowledge about the network also increases with the size of the network.

The paper is organized as follows. Section II discusses related work, whereas Section III describes the threat model we consider. Section IV discusses the identity virtualization mechanism, and the generation of ID pools through hash chains. We present the details of the translation service and update protocol in Sections V and VI respectively. In Section VII, we evaluate our approach with respect to different classes of attacks, and Section VIII reports on the experimental evaluation on a prototype implementation of the mechanism. Finally, some concluding remarks are given in Section IX.

## II. Related Work

Several MTD approaches for mobile and wireless networks have been proposed in the literature, and they are all aimed at dynamically changing a system's parameters in order to increase a malicious observer's uncertainty about its topology and configuration. For instance, [5] proposes a general MTD approach to secure resource constrained distributed devices, and introduces two specific mechanisms for reconfiguring a wireless sensor network at different architectural layers.

As for network-level MTD techniques, several approaches have been proposed for dynamically changing a node's IP address for proactive security. In 2001 Kewley *et al.* [6] presented a technique, called DYNAT (Dynamic Network Address Translation), which is aimed at confusing an adversary sniffing network traffic by obfuscating host identity information in TCP/IP packets entering public segments of the network. Whenever a client wants to communicate with a protected server, the addressing information contained in the header of its request packets is translated (encrypted) by a DYNAT shim before routing the packet to the server. A server gateway receives the packets, reverses the translation in the header fields (decryption) and obtains the true host identity information, used to pass the packets to the target server. Both the client and the server gateway must share a secret seed value, that is used to encrypt the identity information at sender side and decrypt it at the recipient. They are synchronized to periodically change the secret, and thus change the translation results, making it difficult for the adversary to create and maintain a map of the network. Although this technique has the advantage of providing a transparent approach to protect node identities from sniffing, it has been designed to protect a set of static nodes deployed behind a centralized gateway, that represents an interface between the protected network and the external world and performs the translation of addressing information for all incoming and outgoing packets. When considering more complex scenarios, characterized by highly dynamic network configurations, this approach would not work as it might not possible to manage all communications through a centralized gateway and achieve node synchronization.

Atighetchi *et al.* [7] give an overview of the current set of network-level defenses in the DARPA APOD (Application That Participate in Their Own Defense) project. Among the proposed network-centric defense mechanisms, the APOD toolkit also provides a *port and address hopping mechanism*, based on constantly changing a service's TCP identity to both hide the service's real identity and confuse the attacker during reconnaissance. Packets intercepted by attackers will reveal random addresses, which are valid only for a small period of time. For a port attack to be successful, the attacker must discover the current ports and execute the attack all within one refresh cycle. The hopping mechanism is implemented by a client component, directly located on the client machine, that intercepts higher level calls to the real server, and replaces all `(real_address:real_port)` header information with `(fake_address:fake_port)`. The NAT gateway is located either on the server's LAN or directly on the server host and performs the reverse mapping. Even if this approach provides better unpredictability than DYNAT, it also requires synchronization among the two communicating components.

Antonatos *et al.* [8] introduce a proactive defense mechanism called Network Address Space Randomization (NASR) with the objective of hardening networks against worms that use precomputed hitlists of vulnerable targets, by forcing nodes to frequently change their IP addresses. To achieve this goal, the authors implemented an advanced NASR-enabled DHCP server to expire DHCP leases at intervals suitable for effective randomization. As the addresses are changed at the end-points of a communication, active connections are disrupted during the update. Moreover, NASR is limited in the address space as it uses LAN addresses, and requires changes to the end-host operating system, thus making the deployment costly. In [9], the authors introduce an MTD technique called OpenFlow Random Host Mutation (OF-RHM): each host is assigned an address range, selected from the unused network address space, and, at each mutation interval, a virtual IP is chosen from this range. A Software-Defined Networking (SDN) approach is adopted for range allocation and mutation coordination: a centralized controller (NOX) properly installs flows in OpenFlow switches to forward requests and perform the address translation actions.

In summary, all previous techniques rely upon centralized entities to perform ID updates and translation. This make them not suitable for ad-hoc networks, which do not have a fixed infrastructure and need to rely on distributed management. Therefore, our approach differs significantly from previous work, as we propose a distributed approach for MANETs, in which each node builds its own identity pool and is provided with a mechanism to translate virtual IDs into real IDs. In order to allow legitimate nodes to communicate in spite of

frequent ID changes and without synchronization, we introduce an ad-hoc update protocol. Note that, in the type of scenarios considered in most previous approaches, nodes do not need to inform all other nodes about their new identities.

## III. Threat Model

The most commonly adopted threat model for the security analysis of wired and wireless networks was proposed by Dolev and Yao in [10]. The Dolev-Yao model assumes that network nodes adopt a *perfect* cryptographic scheme, such that an encrypted message can only be decrypted by knowing the corresponding decryption key. In this paper, we adopt the Dolev-Yao threat model, and consider attackers able to intercept, spoof and alter any message exchanged among nodes within their hearing range, as well as to inject forged messages or replay old ones.

In our work, we focus on attacks aimed at interfering, steering or eavesdropping normal communications among nodes and we also consider specific attacks against the MTD mechanism and protocols we propose, as they can also become the target of attacks. We do not consider attacks by insiders who know the keying materials and legitimately join all network activities, and focus instead on external attackers. Before launching an attack, an attacker has to scan the network and/or specific mobile nodes in order to collect necessary information for planning the attack. Attacker may adopt different strategies to maximize their gain in terms of collected information. For instance, if the identity space is small enough, they may simply attempt to probe all possible identities (e.g., IP addresses) until a viable exploit is found. Otherwise, they may choose to probe identities observed in recently captured traffic. An attacker may also exchange data with other attackers and perform sophisticated traffic analysis in order to reduce the uncertainty surrounding valid node identities.

In this paper, we do not consider attacks aimed at tracking nodes by analyzing their traffic and mobility patterns and correlating multiple virtual identities previously used by the same node. However, note that node mobility makes it difficult for attackers to correlate multiple virtual identities with a single node. Nevertheless, in order to make traffic analysis even more complex for the attacker and defeat tracking attempts, we may combine the approach proposed here with a mechanism that allows two nodes to switch their respective identity pools when they are within transmission range of each other.

## IV. Identity Virtualization Mechanism

The basic idea behind the proposed MTD mechanism is to use a large number of virtual identities to protect a node's real ID. Each node may have multiple virtual IDs associated with its real ID, and only legitimate nodes should be able to correlate virtual IDs to nodes' real IDs. Virtual IDs are used for communication while real IDs are never publicly used. ID pools can be either pre-loaded on the node or computed at runtime. In this paper, we use hash chain to generate ID pools at runtime.

In order to limit the exposure of a node's ID, and make the IDs an attacker may have collected over time useless, we introduce a *validity interval* for virtual IDs: each ID is used by a node for a limited period of time and then replaced with a different one. To preserve communication among legitimate nodes, we propose a mechanism for legitimate nodes to identify currently valid IDs in the network and determine the mapping between real and virtual IDs. To this aim, we modify and augment the network layer of the protocol stack with:

- a *translation service* for mapping real IDs to virtual IDs and vice versa;
- an *Update Protocol* for disseminating and managing information about nodes' updates.

Information about network status (i.e., current valid IDs) is stored by each node in a *translation table*, and periodically updated through the Update Protocol. This protocol, described in detail in Section VI, has been designed to provide integrity and authentication: it prevents attackers from altering and spoofing protocol messages and also provides a means to counteract replay attacks. In Section VII, we analyze the possible activity of external attackers, aimed at disturbing or steering the protocol, and show its robustness with respect to a variety of attacks.

The translation service is used in conjunction with the routing protocol to handle incoming and outgoing messages, and can map real IDs to virtual IDs by accessing the local *translation table*. When the network layer receives a packet from another node, it uses the translation service to find the real IDs associated with the virtual IDs in the source and destination address fields of the packet. If a match is found, a route is determined based on the local routing table and the packet is broadcast in order to reach the next hop on that route. Similarly, when a node originates a message for a given destination, the network layer uses the translation service to translate the source and destination addresses into valid virtual IDs and uses such IDs for the outgoing message. Moreover, all the control messages the routing protocol needs to exchange to build routes (e.g., AODV requests) use virtual IDs, and are handled as previously described.

### A. Using Hash Chains to Generate ID Pools

Hash chains have been first proposed by Lamport [11] as a password protection scheme against eavesdropping and replay attacks. Since then, they have been employed in a wide range of applications, such as onetime passwords or server supported signatures, due to their interesting properties and low computational costs. In this paper, we use hash chains for generating pools of virtual IDs.

Assume the network is composed of $N$ nodes. For ease of presentation, in the following we assume a node's real identity is simply an integer $i$, with $0 \leq i \leq N-1$. Each node $i$ obtains a shared secret seed $s$ during the join phase – as described in section VI-A – and generates a random initial seed value $x_i$. Then, each node $i$ constructs a hash chain of length $n+1$ by recursively applying a one-way hash function $F$ to the initial seed $x_i$, and combining the argument with $s$ at each step, as shown below, where $F^0(x_i) = x_i$ by default.

$$(\forall k \in [1,n])\ \ ID_i(k) = F^k(x_i) = F(F^{k-1}(x_i), s) \qquad (1)$$

The use of the shared secret $s$ prevents an attacker who has knowledge of the hash function used for generating the hash

chain from logically linking multiple virtual IDs to the same physical node, as described in detail in section VII-A.

Because of the one-way property of the hash function, $F^{n-1}(x_i)$ cannot be generated by knowing the last element $F^n(x_i)$ – called the *commitment* of the chain – without knowing the value of $x_i$. However, knowing $F^n(x_i)$, if $F^{n-1}(x_i)$ is given, its correctness can be verified by checking that

$$F(F^{n-1}(x_i), s) = F^n(x_i) \tag{2}$$

Values in the hash chain of node $i$ will be used as its virtual IDs in the reverse order with respect to generation. In particular, the first virtual ID used by node $i$ will be the commitment of its hash chain, that is $ID_i(n) = F^n(x_i)$, corresponding to hash index $n$. Similarly, the last virtual ID adopted by node $i$ will be $ID_i(1) = F(x_i)$, corresponding to hash index 1.

Any node who observes $ID_i(k)$ is able to compute all the previous virtual IDs already used by the same node, that is $ID_i(j)$, with $k+1 \leq j \leq n$, but no one can compute any of the future virtual IDs, that is $ID_i(j)$, with $1 \leq j \leq k-1$.

We assume the commitments of each node's ID chain are securely distributed to each node. This way, a node receiving a packet from $ID_i(k)$ can use the commitment $ID_i(n)$ to verify the authenticity of the sender by repeatedly applying the hash function $n-k$ times to $ID_i(k)$.

## V. Translation Service

As said earlier, the translation service is integrated into the network layer, without altering the operation of the routing protocol, except that all routing messages will no longer contain real IDs, but virtual IDs provided by the translation service. In other words, our service is orthogonal to routing protocols.

Consider the simple network shown in Figure 1. Each node has a real ID (the number inside the circle) and a current valid virtual ID (the number outside the circle). Suppose that node 1 wants to send a message to node 4 through the routing path 1-2-3-4. Figure 2 shows what happens at the sender node: the message is processed at the routing level in order to find the path towards the destination (based on the adopted routing protocol). The routing protocol then invokes the translation service, which translates the source and destination IDs to their corresponding currently valid virtual addresses, based on the local *translation table*. Finally, the message is broadcast and reaches the next hop designated by the routing protocol.

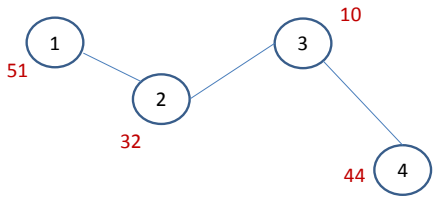


Fig. 1. A simple network

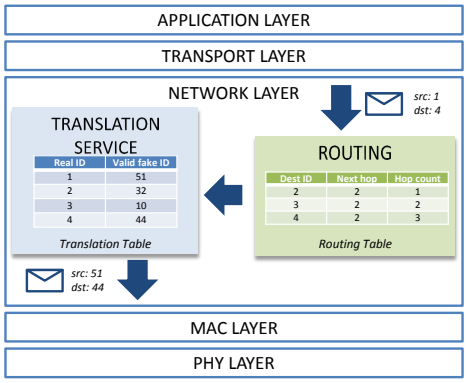


Fig. 2. Resolving IDs at the originator node (node 1)

The process is similar at any intermediate router node. As shown in figure 3, an intermediate node receiving a packet, let's say node 2, will first translate the virtual IDs to the real IDs in order to find the correct route towards the destination. Then, it will forward the packet to the correct next hop (node 3), but still using virtual IDs.

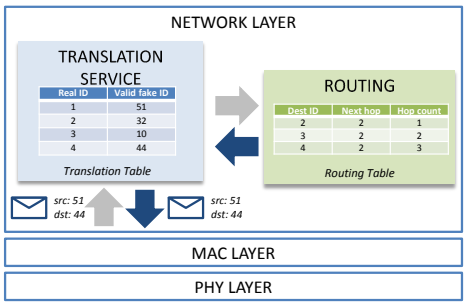


Fig. 3. Resolving IDs at an intermediate node (node 2)

As virtual IDs change periodically, a malicious observer is presented over time with many node identifiers, even related to the same data flow. In normal conditions, the attacker could collect information about node IDs observed in the network, and perform some vulnerability scan on those nodes in order to discover software weaknesses. She could then exploit such vulnerabilities to launch specific attacks against the nodes. With the proposed approach, by the time the attacker has collected enough information to launch the attack against a node, the ID may have changed, thus thwarting the attack itself.

## VI. Update Protocol

Two fundamental concerns must be taken into account when designing the Update Protocol: (i) when should nodes change their IDs?, and (ii) how can legitimate nodes know how to communicate with one another when their public identities keep changing over time?

As for the first problem, several strategies can be adopted: in the simplest scenario, nodes could decide to update their IDs once a shared timer expires. This strategy does not require to exchange control messages over the network and allows each node to know exactly the current valid ID of all

other nodes at any given time, but it also relies upon strict clocks' synchronization, which can become difficult to achieve efficiently when the network includes several hundreds nodes. We argue that the overhead can be better controlled if each node could autonomously and asynchronously decide when to update its virtual ID and communicate its decision to the other nodes through ad-hoc UPDATE messages. Therefore, we assume that each node $i$ updates its ID when a local timer expires. Such timer is randomly selected in the interval $[T_{min}, T_{max}]$, where $T_{max}$ is the maximum ID validity interval allowed (i.e., a validity interval larger than $T_{max}$ makes the MTD mechanism ineffective), and $T_{min}$ is the minimum ID validity interval allowed (i.e., a validity interval smaller than $T_{min}$ would not give enough time for an UPDATE message to propagate across the network before the next update is triggered).

A virtual ID $ID_i(k)$ is used by node $i$ only within a *validity interval* $\Delta T_i(k)$. When the timer associated with such validity interval expires, node $i$ will replace its current virtual ID – $ID_i(k)$ – with the next ID in the ID chain – $ID_i(k-1)$ – and randomly choose the duration of the next validity interval $\Delta T_i(k-1)$.

In order to preserve network communication, as previously said, an UPDATE message is broadcast (flooded) by the updating node $i$. In particular, when node $i$ replaces $ID_i(k)$ with $ID_i(k-1)$, it will generate an UPDATE packet containing:

- the new valid ID $ID_i(k-1)$ as its source ID,
- the hash index $(k-1)$, corresponding to the new ID in node $i$'s hash chain.

The entry for node $i$ in node $j$'s translation table looks as shown in Table I.

TABLE I. ENTRY FOR NODE $i$ IN THE TRANSLATION TABLE OF NODE $j$

| NodeID | HashIndex | CurrentID |
|---|---|---|
| $i$ | $k$ | $ID_i(k)$ |

$NodeID(i)$ represents the real ID of the node, while $CurrentID(i)$ and $HashIndex(i)$ are respectively the current valid virtual ID for node $i$ and the hash index of such ID in node $i$'s hash chain.

Assume that node $j$ receives an UPDATE containing $ID_t(h)$ as the source ID and $h$ as the hash index. For each entry $i$ in the local translation table, if $HashIndex(i) > h$ node $j$ computes $Q = F^{HashIndex(i)-h}(ID_t(h))$, and checks if $Q = CurrentID(i)$. If this equality holds, it means that node $i$ is the originator of the UPDATE message, therefore node $j$ will update the corresponding entry in the translation table by setting: $CurrentID(i) = ID_t(h)$ and $HashIndex(i) = h$.

### A. Joining and leaving the network

A legitimate node must be able to join and leave the network at any time. In order to allow for correct communication in a completely distributed scenario, a joining node must send a valid request to the network. Clearly, a node's join or leave request must be authenticated. Achieving nodes' authentication is a non-trivial task in MANETs, as they typically lack a fixed infrastructure or centralized management. In this section, we assume that legitimate nodes are provided with two shared secrets: $k$ is used to encrypt request packets such that only legitimate nodes are able to correctly handle them, while $s$ is a shared secret seed used by each node to alter the argument of the hash function, as discussed in section IV-A.

**Join.** As shown in section VI, in order to allow for correct communication, legitimate nodes need to share the commitment of their hash chains. Even if it is hardly likely, two or more different nodes could choose the same initial seed, resulting in identical hash chains. The problem is very similar to the Address Assignment Problem in MANETs, which has been addressed by several researchers [12], [13]. The Zeroconf working group [14] proposed a mechanism [15] to allow nodes to auto-configure their addresses: when a node joins the network, it randomly chooses an IP address and sends an ARP (Address Resolution Protocol) message for the chosen IP address. If the IP address is already used, the new node is informed, and chooses another address and restarts the procedure. If the new node receives no response within a given timeout, it concludes that the IP is available so it can use it.

We use a similar mechanism to address the problem: when a node wants to join the network, it chooses an ID $i$ and the initial random seed $x_i$. Then it computes the hash chain commitment $ID_i(n)$. The commitment is used as the source address in a JOIN REQUEST packet, whose payload is composed of the ID $i$ and a random number $r_i$. The packet is encrypted with the shared secret $k$ and flooded through the network.

If a node recognizes the ID and/or the commitment in the JOIN REQUEST packet as its own ID and/or commitment – after decrypting it with the shared secret – it broadcasts (floods) a JOIN RESPONSE packet, meaning that the ID and/or commitment in question are already in use. If the joining node does not receive a JOIN RESPONSE packet before a local timer expires, it assumes the ID and the commitment are available, and uses them to join the network.

As said, a node receiving the JOIN REQUEST packet, compares the ID and commitment contained in the packet with its own ID and commitment. If they are different, the packet is queued and considered as "pending" until a proper timeout expires. If no JOIN RESPONSE packets are received in this time interval regarding that ID or commitment, the node assumes that the ID-commitment pair belongs to a new legitimate network node, and updates its own translation table accordingly. If a JOIN RESPONSE packet is received for a pending ID-commitment pair, it is removed from the queue.

Two or more nodes could issue a JOIN REQUEST for the same commitment, or the same ID, or both. In order to distinguish between its own request (that has been rebroadcast by neighbors) and the request of a different node when they contain the same ID-commitment pair, each requesting node compares its randomly generated number $r_i$ with that contained in the request packet. If they do not match, the node sends a JOIN RESPONSE packet, and chooses a new random ID and a new secret, and starts the join procedure again.

**Leave.** Several mechanisms are already available, at the routing level, to detect a node's departure based on link breaks. Without any additional protection, an attacker may be able to identify when a node is leaving the network and determine

its last used ID. As the leaving node will no longer issue UPDATEs, such ID will always be valid for other legitimate nodes, and an attacker may able to use it to establish communications with them. Even if the attacker's knowledge of the network is still limited, in order to mitigate this risk, we can introduce a timer associated with each entry in the translation table. As all nodes update their ID at least every $T_{max}$ time units, if no UPDATE packets are received for a given node after this timer expires, any additional data packet received from that ID is considered a sign of a malicious activity. In this way, we can avoid to perform a network flooding also for the leave procedure, thus saving nodes' energy.

### B. Nodes re-initialization

As discussed, each node $i$ has $n$ distinct IDs in its pool. Once this pool has been depleted, a *re-initialization* procedure must be performed. The re-initialization procedure consists in choosing a new secret $x_i$, generating a new ID pool accordingly, and distributing the corresponding commitment to the entire network. This can be achieved in the same way as the join procedure described above.

### C. Overhead and latency

Clearly, the higher level of security achieved by the periodic UPDATE process is paid with the introduction of computational overhead and latency. The described procedure involves a linear look-up in the local translation table, and the recursive application of the hash function to find a match. Indeed, even if the computational cost of the hash function is very small, if the network is composed of thousands of nodes, the linear search can introduce a delay in the handling of UPDATE packets. Actually, if the update frequency is not too high, as not all nodes perform updates at the same time, this delay can be easily absorbed. Moreover, a simple strategy can be adopted to reduce the impact of repeated UPDATE packets, either received from legitimate neighbors due to the flooding mechanism, or deliberately replayed by malicious nodes: in fact, each time an UPDATE packet is received, the recipient node has to verify the match with all the entries of the table, thus wasting time and resources before making the decision of dropping the packet. In order to avoid this, every time a node successfully verifies the content of an UPDATE packet and updates its translation table accordingly, it stores the ID contained in the packet in an *Update Cache Queue*. The queue contains at most $M$ entries, with $M < N$: when the queue is full, the next value will be added to the tail, and the head will be removed. When an UPDATE packet is received, the node will first search in the *Update Cache Queue* for the source ID contained in it. If an entry is found, the packet is automatically discarded, otherwise it is processed normally.

When a node $i$ replaces its current valid ID, its UPDATE packet will take some time to reach all other nodes of the network. During this time interval, that we call *Update Latency*, legitimate nodes could have a different view of the network status, and such inconsistency could affect network traffic in which the updating node is involved. In particular, some packets traveling over the network will contain the old valid ID of node $i$ in the source or destination fields. Each node $j$ that receives one of such packets will process or discard them depending on whether or not it has already received the UPDATE from $i$.

If the communication is carried out over a reliable transport protocol such as TCP, each message sent by a node must be acknowledged by the recipient. If the sender does not receive an acknowledgment (ACK) within a given amount of time, it will retransmit the message – with a certain frequency – until an ACK is received. This behavior could lead to a deadlock condition when the recipient of a transmission updates its ID during the transmission. Assume that node $i$ is sending messages to $j$ and that, at a certain time, $j$ updates its current ID. Until the UPDATE packet of $j$ is received by $i$, the messages sent by $i$ will contain an invalid ID in the destination field, and thus will be dropped. Consequently, node $i$ will start retransmitting these messages waiting for the relative ACKs. If no further mechanism is provided, the ACKs for these messages will never be received by the source node, as the messages sent by $i$ to the invalid ID will be discarded automatically by the translation service, and the messages will be retransmitted over and over again. In order to cope with this problem, the transport layer should have access to the translation table, in order to check the current valid ID associated with the recipient, and update the destination field before retransmitting a queued packet.

Note that no deadlock condition can occur when the sender updates its ID during a transmission. Assume that node $i$ is sending messages to $j$ and that, at a certain time, $i$ updates its current ID, continuing to send messages with the new ID; also assume that the UPDATE packet of node $i$ is in some way delayed, so that the data packet arrives to the recipient $j$ before the UPDATE. In this case, $j$ will discard the message (as it contains an invalid ID) and any other retransmission of that message until it gets the UPDATE. At this point, $j$ will recognize the ID of the sender as valid and send back an ACK to $i$, that will stop retransmitting the message.

If an unreliable transport layer is used instead, such as UDP, no retransmissions will be issued and the packets containing not valid IDs will be simply discarded.

The following subsection provides further discussion about the management of UPDATEs' losses and delays.

### D. Managing UPDATE losses and delays

In the presence of network congestion and/or node mobility, it is likely to experience occasional losses or delays of UPDATE messages, which may cause inconsistencies among different nodes' view of the status of the network. In particular, as shown in the previous section, while the UPDATE protocol itself is not influenced by UPDATE losses, as each UPDATE message can be handled independently of its logical predecessors, application or routing traffic could be affected if involved nodes do not share the same information about network status.

Consider the case of a mobile network in which node $i$ and node $j$ are communicating. Assume that at time $t$ the network becomes partitioned in two subnetworks, one containing $i$ (partition $A$) and the other containing $j$ (partition $B$), and that node $i$ issues an UPDATE just after the partitioning, so that nodes in partition $B$ do not receive the UPDATE. Without the ID update mechanism, if partitions did merge at

time $t + \delta$, only packets sent during interval $\delta$ would be lost, and communication could be established again without further losses. With the ID update mechanism instead, also all packets sent after merging and before the next update (assuming this is correctly received by all nodes) would be lost, as no one of the nodes belonging to partition $B$ is able to find a mapping for the new ID of node $i$. In order to mitigate this problem, in the presence of mobility, a node could decide to reply the last UPDATE during a validity interval, to help synchronization. Of course, a trade off exists between replay frequency and overhead introduced.

## VII. Security Evaluation

Periodically changing a system's configuration to augment security is an intuitive principle. Nevertheless, while several approaches have been proposed in the literature to evaluate the security provided by a system [16], [17], there is still a lack of metrics to quantitatively evaluate the benefits of reconfiguration, that is the *security gain* resulting from the application of MTD strategies. The work presented in [5] represents a first step towards quantifying the benefits of Moving Target Defense.

In this section, to demonstrate the robustness of the proposed approach, we present a detailed qualitative evaluation of its security with respect to a variety of attacks. In the previous sections, we outlined the primary benefits of periodically changing a node's ID. Here, we address both specific attacks against the update protocol itself, and some of the most common routing attacks, showing how our protocol's design is able to thwart and/or mitigate them.

We only consider external attackers, that do not know a node's sensitive information, such as the random seed used to generate the hash chain and the shared secret used to encrypt JOIN REQUEST and JOIN RESPONSE packets.

### A. Attacks against the UPDATE protocol

With respect to the UPDATE protocol itself, an attacker could perform different actions targeted at jeopardizing a specific phase of the protocol:

**JOIN phase**. As attackers do not know the shared secret, they cannot pretend to be legitimate nodes and join the network. For the same reason, also attacks aimed at forging JOIN REQUEST packets to verify if some identities exist in the network (by verifying whether a JOIN RESPONSE packet is received for those identities) are not possible.

**UPDATE phase**. An attacker may try to forge an UPDATE packet with its own ID or replay an intercepted UPDATE packet to pretend that a node is changing its ID. However, neither of these attacks can succeed. In fact, these packets will be respectively recognized as invalid or old and discarded by legitimate nodes. Indeed, the attacker could use this strategy to perform a denial of service attack, as legitimate nodes are forced to process each packet before discarding it. The solution proposed in section VI-C can mitigate this risk.

**ID generation phase**. Assuming the hash function used in the generation of ID pools is known to attackers, they could perform the type of brute force attack described in the following, if provided with significant processing power. An attacker could systematically test every seed in the seed space and generate the entire hash chain starting from that seed, and check whether the generated commitment corresponds to one of the previously gathered commitments. Clearly, a large seed space would thwart brute force attacks.

### B. Attacks against the routing protocol

In this section we consider the most common routing attacks [18], and show how our protocol's design is able to thwart and/or mitigate many of them. We assume the adopted routing protocol is AODV, properly modified as described in section V.

**Blackhole attack**. The blackhole attack consists in generating incorrect routes so that packets are no longer forwarded to the proper recipient but instead get lost or are redirected to the attacker itself. In our modified version of AODV, routes are determined by exchanging AODV RREQ and RREP packets containing valid virtual IDs instead of real IDs. In order to advertise itself as having a valid route to a destination node and intercept all traffic towards it, a malicious node should have a valid virtual ID. Since the attacker does not have a valid virtual ID, this attack can be prevented.

**Wormhole attack**. In the wormhole attack, an attacker records packets at one location in the network and tunnels them to another location, thus preventing for example the discovery of any routes other than through the wormhole. Similarly to Blackhole attacks, in order to re-route traffic through itself, a malicious node needs a valid virtual ID, therefore this attack is not feasible.

**Sybil attack**. A malicious node can illegitimately take on multiple identities [4]. However, as the attacker cannot legitimately join the network, such identities will be ignored by legitimate nodes.

**Routing message flooding attack**. In this type of attack, attackers do not follow the specifications of the routing protocol. As an example, an attacker can originate many AODV REQUEST packets using some recently heard IDs and flood the network.

As legitimate nodes periodically change their IDs, the spoofed ID used for malicious packets may no longer be valid. In this case, they will be ignored, thus mitigating the attack. Moreover, specific solutions aimed at addressing this problem have been proposed [19].

Another type of routing message flooding attack is the Routing Table Overflow attack, in which the attacker advertises routes to non-existent nodes to generate overflow in the routing table. As there is no valid virtual ID associated with these nodes, the attack cannot be performed.

**Route invalidation attack**. In this attack, a malicious node could forge ERROR messages to invalidate routes, using recently overheard virtual IDs. Even in this case, as legitimate nodes periodically change their IDs, the spoofed ID used in the malicious packet may no longer be valid, thus mitigating the attack.

## VIII. Experimental Evaluation

As discussed in Section VI-C, the higher level of security achieved through the periodic update process is paid with

the introduction of computational overhead and latency. The overhead due to the look-up operation in the local translation table – performed by each node every time a packet is processed – and the update operations themselves affect normal communications by slowing them down or by increasing the number of retransmissions. Due to the update latency, some nodes can temporarily hold a different view of the network status. Routing or data traffic involving such nodes in this time interval could be negatively affected, as packets containing non valid IDs will be automatically discarded. This can lead to delays and increased retransmissions in reliable networks, or packet losses in unreliable networks.

In the following, we present a set of experiments aimed at evaluating the performance of the proposed MTD mechanism in terms of overhead. We implemented the proposed mechanism in NS-2, by developing an agent for running the UPDATE protocol and managing the translation service. The nodes of the simulated network run TCP on top of a modified version of AODV, which communicates with the translation service according to the described design.

In our experiments, we set the simulation time to 200 seconds and considered validity intervals of decreasing length. Each node randomly chooses a timer in a given time interval $[T_{min}, T_{max}]$ and uses it as the duration of the current validity interval. Table II shows the considered time intervals and the corresponding *update frequency*, given by the number of update operations performed by each node during the simulation time.

TABLE II. VALIDITY INTERVALS CONSIDERED IN THE EXPERIMENTS

| $(T_{min}, T_{max})$ | Update frequency |
|---|---|
| (100,105) | 1 |
| (50,55) | 3 |
| (20,25) | 9 |
| (10,15) | 19 |

For each value of the update frequency, we generated 10 different random scenarios and recorded several statistics, such as the number of nodes, the traffic patterns, and changes in the nodes' mobility patterns.

Figure 4 shows how the number of retransmitted TCP packets increases when the update frequency increases, in networks composed of 100, 500 and 1,000 nodes respectively, with a single-sender/single-receiver TCP communication pattern. As shown, when reducing the validity interval, the rate at which the percentage of retransmitted packets increases, becomes greater. This trend is clearer when considering a larger number of nodes.

As said earlier, due to the update latency, some of the packets exchanged over the network will be dropped as they use invalid IDs. Figure 5 shows the percentage of *well-formed* received packets, that is packets that are correctly processed by recipient nodes, as they contain valid IDs. Even in this case, the higher the update frequency, the faster the resulting percentage of well-formed packets decreases.

Figure 6 shows the total number of UPDATE packets traveling over the network during the simulation time, which provides a measure of the packet overhead introduced by the UPDATE protocol. The number of UPDATE packets transmitted on the network is quadratic in the number of nodes.

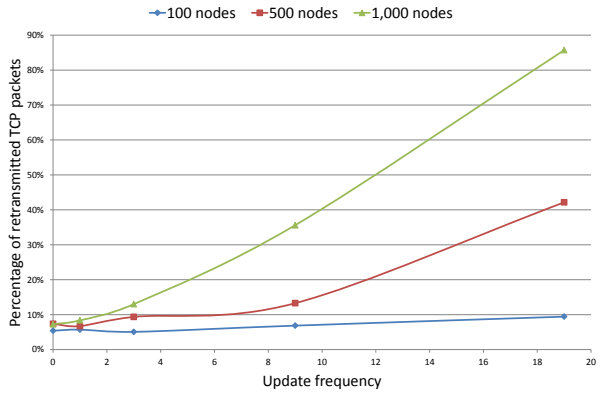


Fig. 4. Percentage of retransmitted TCP packets vs. update frequency

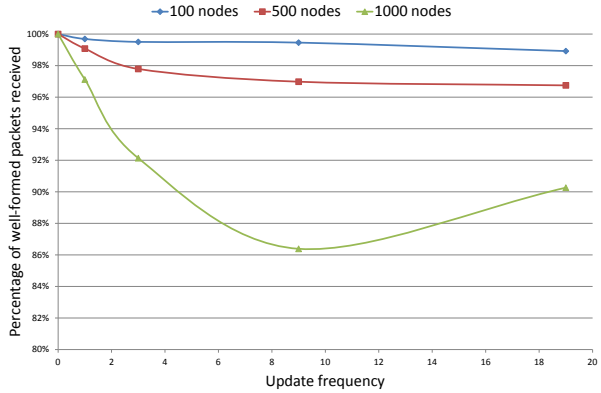


Fig. 5. Percentage of well-formed packets vs. update frequency

As expected, a trade-off exists between the level of security provided by the update mechanism and the resulting overhead/performance. As shown in the charts included in this section, when the size of the network increases, performance rapidly degrades and overhead increases as the validity interval becomes smaller. This suggests that larger validity intervals should be chosen for larger networks in order to limit the overhead. Indeed, the effort required from an attacker to gain knowledge about the network is proportional to the number of nodes, therefore it is reasonable to reduce the update frequency in large networks, while retaining the security benefits of the proposed mechanism.

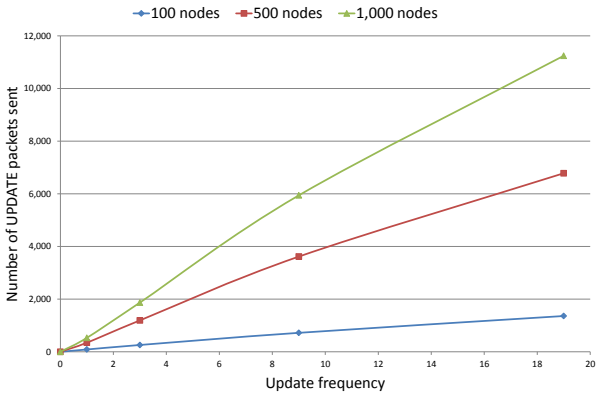


Fig. 6. Total UPDATE packets sent over the network vs. update frequency

Figure 7 shows the percentage of well-formed received packets, in the case of 10 concurrent TCP connections and of a single TCP connection respectively, for a network composed of 100 nodes. Clearly, as more packets travel through the network, the percentage of discarded packets is greater in the case of multiple connections.

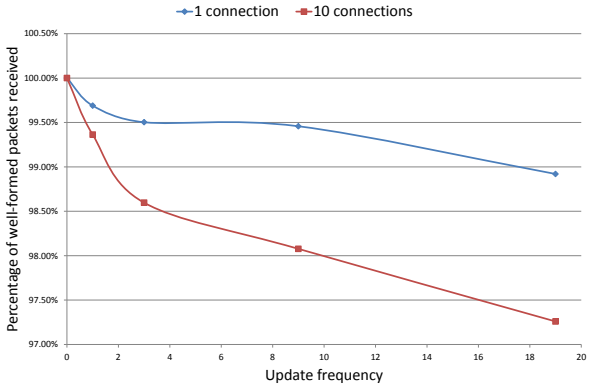


Fig. 7. Percentage of well-formed packets for single and multiple connections

Finally, Figure 8 shows the percentage of well-formed packet for a network composed of 100 nodes, in the case of multiple connections and different moving speeds.

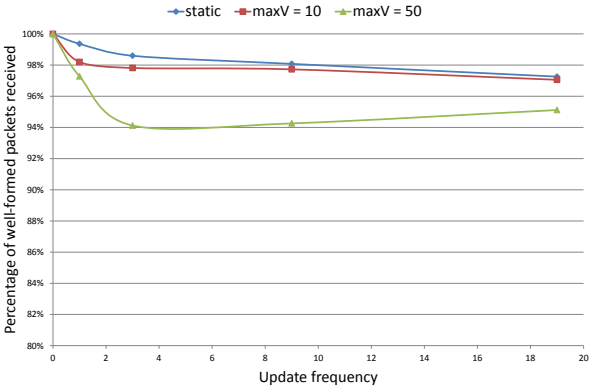


Fig. 8. Percentage of well-formed packets as nodes' speed increases

## IX. Conclusions

Mechanisms for continuously changing or shifting a system's attack surface are emerging as game-changers in cyber security. Such mechanisms increase the complexity for attackers, limit the exposure of vulnerabilities, and increase overall system resiliency. In this paper, we proposed a novel MTD mechanism for periodically changing the virtual identity of nodes in a MANET. The proposed mechanism turns a classical attack mechanism – Sybil attack – into an effective defense mechanism, with legitimate nodes periodically changing their virtual identity in order to defeat the attacker's reconnaissance efforts. In order to preserve the ability for legitimate nodes to communicate, we modified the network layer by introducing (i) a *translation service* that can map virtual identities to real identities; (ii) a protocol for propagating updates of a node's virtual identity to all legitimate nodes; and (iii) an ad-hoc mechanism for legitimate nodes to securely join the network. We showed that the proposed approach is robust, and can prevent or mitigate different types of attacks. We also showed that the overhead introduced by the update protocol is low.

## References


[1] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, ser. Advances in Information Security. Springer, 2011, vol. 54.

[2] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, May 2011.

[3] Executive Office of the President, National Science and Technology Council, "Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program," http://www.whitehouse.gov/, December 2011.

[4] J. R. Douceur, "The sybil attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., vol. 2429, 2002, pp. 251–260.

[5] V. Casola, A. D. Benedictis, and M. Albanese, "A moving target defense approach for protecting resource-constrained distributed devices," in *Proceedings of the 14th International Conference on Information Reuse and Integration (IEEE IRI 2013)*, 2013.

[6] D. Kewley, R. Fink, J. Lowry, and M. Dean, "Dynamic approaches to thwart adversary intelligence gathering," in *Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX 2011)*, vol. 1, 2011, pp. 176–185.

[7] M. Atighetchi, P. Pal, F. Webber, and C. Jones, "Adaptive use of network-centric mechanisms in cyber-defense," in *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2003)*, May 2003, pp. 183–192.

[8] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," *Computer Networks*, vol. 51, no. 12, pp. 3471–3490, August 2007.

[9] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," in *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2012)*, 2012, pp. 127–132.

[10] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, March 1983.

[11] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, November 1981.

[12] M. Mohsin and R. Prakash, "Ip address assignment in a mobile ad hoc network," in *Proceedings of the Military Communications Conference (MILCOM 2002)*, vol. 2, 2002, pp. 856–861.

[13] S. Nesargi and R. Prakash, "MANETconf: Configuration of hosts in a mobile ad hoc network," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2002)*, vol. 2, 2002, pp. 1059–1068.

[14] The Zero Configuration Networking (Zeroconf) Working Group, http://www.zeroconf.org/.

[15] S. Cheshire, B. Aboba, and E. Guttman, "Ietf rfc 3927: Dynamic configuration of ipv4 link-local addresses," http://www.ietf.org/rfc/rfc3927.txt, May 2005.

[16] V. Casola, A. Mazzeo, N. Mazzocca, and V. Vittorini, "A policy-based methodology for security evaluation: A security metric for public key infrastructures," *Journal of Computer Security*, vol. 15, no. 2, pp. 197–229, April 2007.

[17] V. Casola, R. Preziosi, M. Rak, and L. Troiano, "A reference model for security level evaluation: Policy and fuzzy techniques," *Journal of Universal Computer Science*, vol. 11, no. 1, pp. 150–174, 2005.

[18] D. P and A. Kannammal, "Security attacks and defensive measures for routing mechanisms in manets - a survey," *International Journal of Computer Applications*, vol. 42, no. 4, pp. 27–32, March 2012.

[19] P. Yi, Z. Dai, S. Zhang, and Y. Zhong, "A new routing attack in mobile ad hoc networks," *International Journal of Information Technology*, vol. 11, no. 2, pp. 83–94, 2005.