Minimal Trusted Computing Base for MANET Nodes

Vinay Thotakura, Mahalingam Ramkumar
Department of Computer Science and Engineering
Mississippi State University, MS.

***Abstract*—Securing any MANET routing protocol requires measures to ensure that routing information advertised by a node (*to* its neighbors) is consistent with routing information assimilated by a node (*from* its neighbors). We investigate a minimal trusted computing base (TCB) for MANET nodes to ensure this requirement. We outline low complexity TCB functions which can be executed inside trustworthy boundaries of resource limited trustworthy MANET modules (TMM). The TCB functions are used to maintain a table of active neighbors, create routing records (RR), authenticate RRs to TMMs in neighboring nodes, receive authenticated RRs, and update RRs, subject to simple rules. Even while the dynamic RR database of every node is stored outside the TMM, by storing the root of an index ordered Merkle hash tree (IOMT) the TMMs can ensure that nodes cannot modify, replay or even hide RRs.**

I. Introduction

Nodes forming mobile ad hoc networks (MANET) do not depend on an infrastructure of dedicated routers. The nodes themselves share the responsibility of routing packets between nodes, by adhering to a MANET routing protocol. In all routing protocols participants broadcast routing information to nodes within their range: either periodically, or on demand. In general, a routing packet broadcast by a node A provides two kinds of information to neighbors of A. viz., i) first-hand information about A - that A is still a neighbor, or has entered the neighborhood; and ii) n^{th}hand information regarding other nodes.

Based on information provided by A, (for example) that a node D is 5 hops away from A, neighbors of A who do not have currently have a path to D that is less than 6 hops, add this information regarding D in a database of routing records (RR), indexed by the destination identity D. On receiving a routing packet from a neighbor, a node updates its RR database, and may advertise updated information to its neighbors.

The need to rely on indirect (second-hand, third-hand, etc.) information renders MANET protocols highly vulnerable to attacks. Securing MANET routing protocols demands tangible measures to ensure that the *RRs advertised by a node are consistent with RRs assimilated and stored by a node.*

A. Trusted Computing Base

For any computing system with a desired set of assurances, the trusted computing base (TCB) "is a small amount of software and hardware we need to rely on" [1] to realize the desired assurances. More specifically, as long as the TCB is worthy of trust, the desired assurances can be realized even if all *other* components of the system misbehave.

In typical approaches for securing MANET protocols the TCB includes a set of cryptographic algorithms which are assumed to be unbreakable, some trusted entities like key distribution centers and/or certificate authorities who are trusted to perform their tasks, and the notion that the keys assigned to nodes will not be privy to unintended parties. This TCB is amplified using security protocols which specify policies for cryptographic authentication/verification of routing data, with the goal of realizing the desired assurances.

Unfortunately, leveraging this TCB to ensure consistency of assimilated and advertised RRs can be i) expensive due to high overhead for the security protocol, or ii) even infeasible. Furthermore, some components of the TCB may not actually be *worthy* of trust. For example, while it is assumed that the secrets assigned to a node A are privy only to node A, in practice nodes may not protect their secrets well. Some nodes may even collude together. There are numerous ways in which a set of n colluding nodes [2], all of which have access to each other's keys, can wreak havoc on the subnet.

B. Efficient TCB for MANET Nodes

One possible approach to expand the TCB is to implement some special TCB functionality inside trustworthy boundaries like tamper-responsive computing modules. To merit trust, and to ensure that modules that offer this functionality can be realized at a low cost, the TCB functions should be as simple as possible. The contribution of this paper stems from the question

> What is a minimal TCB for MANET nodes to ensure consistency between assimilated and advertised topology information?

The broad goal of the TCB is to prevent nodes from acting in a malicious fashion. The challenge is in minimizing the storage and computational requirements for the tasks performed inside the trustworthy boundary. In the proposed model, a low complexity trustworthy MANET module (TMM) housed in every MANET node offers a set of TCB functions, which can be leveraged with low overhead, to ensure that nodes can only advertise RRs consistent with assimilated RRs.

In this approach every TMM protects some secrets that are hidden even from the node which houses the TMM. Using their respective secrets, any two TMMs X and Y (housed in nodes X and Y respectively) can compute a pairwise secret K_{XY}. Routing records are created by TMMs, modified by TMMs, and exchanged between TMMs (authenticated using pairwise secrets). The TMMs also maintain a table of neighbors. RRs are authenticated only to neighbors, and consequently TMMs accept RRs only from TMMs of neighbors. From the perspective of a TMMs, a nodes which houses the TMM is an untrusted middle-men, who

1) has complete control over the communication interfaces and channels;
2) has physical access to the TMM, but cannot expose secrets protected by the TMM, or modify the TMM's TCB functionality; and
3) stores routing records.

The rules that govern how and when RRs should be modified, deleted or advertised are not complex enough to pose significant challenges for implementation of such rules inside resource limited TMMs. The primary challenge stems from the fact that the dynamic database of RRs cannot be stored inside the TMM. In our approach, a novel construct, an "index ordered Merkle-hash-tree" (IOMT) is employed to ensure that "storing a single value (the root of the tree) inside the TMM, and storing an entire dynamic database of RRs outside the trusted boundary of TMM, is for all practical purposes, equivalent to storing the entire database of RRs inside the TMM." By storing the root of the tree, and performing simple sequences of cryptographic hash and logical operations, TMMs can ensure that RRs stored by untrusted nodes cannot be modified, replayed, or even hidden from the TMM.

We refer to the functionality built into TMM to i) maintain a neighbor table; and ii) maintain the RR database, viz., accept, process, store and delete RRs subject to the rules governed by the routing protocol, as MANET TCB functions. In this paper we provide an outline of simple MANET TCB functions, which can be implemented inside low complexity TMMs that can be realized at a low cost. More specifically, such TCB functions are simple enough to be implemented using a fixed sequence of logical and cryptographic hash operations, and can be leveraged with low overhead to ensure consistency between assimilated and advertised topology information.

In Section II we begin with a broad overview of MANET routing, a high level description of attacks on MANET routing protocols, and current approaches to address attacks. In Section (III) we introduce index-ordered Merkle trees (IOMT). In Section IV we begin with an overview of TMMs, and discuss three simple TCB functions for MANET nodes. In Section IV-E we outline how the TCB functions can be leveraged by nodes adhering to the ad hoc on demand distance vector (AODV) protocol. Conclusions are offered in Section V.

II. MANET Routing Protocols

MANET routing protocols can be classified into proactive and reactive protocols. In proactive approaches like dynamic sequenced distance vector (DSDV) [5] protocol, nodes strive to maintain a consistent view of the topology of a connected subnet at all times. In reactive protocols like the ad hoc on-demand distance vector (AODV) [6] protocol and the dynamic source routing (DSR) [7] protocol, routes are determined on demand.

In DSDV every node maintains a table indicating all nodes in the subnet, a distance metric to each node, a sequence number associated with the metric, and the neighbor which provided this information (which is the next hop to reach that destination). Every node periodically advertises the distance metrics (hop count) for all nodes in the subnet to all neighbors. Based on the information received from all neighbors, a node recomputes its shortest metric to all other nodes in the subnet. This updated information is advertised in the next update provided to neighbors.

In reactive protocols, discovery of routes starts with a query in the form of a route request (RREQ), which includes a fresh sequence number of the initiator. Any node receiving the RREQ gains knowledge of a mechanism to reach the initiator of the RREQ. A node processing the RREQ responds to the query by i) sending a route response (RREP) packet if it has knowledge of a path (DSR) or forwarding information (in AODV) to the desired destination; or ii) forwarding the RREQ onwards otherwise. After a route is established intermediate nodes may invoke route error (RERR) messages if a link required to maintain the route is broken.

Some protocols like the temporally ordered routing algorithm (TORA) [8] can operate in both proactive and reactive modes at the same time. Nodes that desire to be reached can proactively (and periodically) instantiate a process of flooding an OPT packet with a fresh sequence number. A node that does not have a path to a destination can also send a query (QRY) packet, in response to which it receives an UPD packet (from a node which does have a path to the destination) indicating the distance to the desired destination.

A. Attacks on MANET Routing

Attacks on MANET routing protocols can be classified into passive and active attacks. Passive attackers do not transmit any data. The intention of such attacks could be to merely observe traffic patterns or snoop on data exchanged between nodes. Selective participation, where (for example) a node announces its presence only if it needs to send or receive data packets, but remains silent otherwise (does not take full part in routing), can also be considered as a passive attack.

Active attacks are performed by taking an active part in relaying messages and routing information. Such attacks attempt to modify relayed routing data or replay old routing data (which has been invalidated due to topology changes), with the intention of disrupting the routing fabric. Some specific instances of active attacks include modification of hop counts in routing tables, reporting of fictitious neighbors, or randomly corrupting routing information / data packets that are relayed.

The specific intention behind such attacks can vary. For example, an active attacker may shorten the hop count in order to ensure that most packets pass through the attacker. The attacker can then suddenly start dropping packets, or send supercilious RERR messages to create large scale disruptions in the subnet. On the other hand, an attacker may increase the hop count to reduce the chance of being in the route to the destination, thereby shirking responsibility for relaying data.

Active attacks also include rushing attacks, where a rushed bad packet is intended to preempt other good packets, and thereby lower the chance of successful establishment of routes. Other forms of active attacks include simple denial of service (DoS) attacks where the attacker attempts to engage nodes in performing fruitless tasks - for example, by sending random bits in place of a digital signature, causing victims to invest substantial effort to verify the "signature." Such attacks can be especially severe in scenarios involving resource limited devices.

1) Addressing Attacks: Secure MANET routing protocols include explicit features to address various attacks. In general, mechanisms to address passive attacks require nodes to monitor their neighbors to determine which of neighbors did or did not send packets. For this purpose, a node processing a packet sent by "node A" needs to verify that the packet was sent by A. This is made possible through a cryptographic authentication token accompanying the packet. Strategies for addressing active attacks strive to detect inconsistencies in the routing information. For this purpose, verification of multiple cryptographic tokens (created by multiple of nodes) is called for [9] (for example, verification of authentication tokens appended at two hops, or even all hops).

Most often, due to the overhead for providing assurances several "clever" optimizations in routing protocols (which can improve the overall efficiency of the protocol) are dissuaded in secure extensions of such protocols. For example, in DSR a node A can receive or overhear various RREQ/RREP packets that indicate segments of paths which include A or any neighbor of A. While using this knowledge to create responses for queries from other nodes can lower the overhead and reduce latency, providing assurances of the integrity of such responses is substantially more difficult. For this reason, secure extensions of DSR typically allow only RREP by the final destination [9], [2]. In both DSR and AODV, once a route has been established, link failures can cause the path to break down. While permitting intermediate nodes to make local decisions to re-route packets can reduce the overhead for finding alternate routes, most secure extensions will require the source to re-initiate the RREQ process [10].

III. Index Ordered Merkle Tree

A binary Merkle-tree [11] of height L with $N = 2^L$ leaves $l_0 \cdots l_{N-1}$ can be can be constructed using two one-way functions - both of which can be derived from a cryptographic hash function $h()$ (for example, SHA-1). A function $v_i = h_l(l_i)$ maps a leaf to an intermediate node v_i at height 0; a function $z = h_n(x, y)$ maps two intermediate nodes x and y at height $n-1$ to a node z at height n. For example, $z = h(x \parallel y)$

Figure 1 depicts a Merkle tree with $N = 16$ leaves of height $L = 4$. The root of the tree is a commitment to all leaves. To prove to an entity (who only has access to the root of the tree) that a leaf l_i is part of the tree, the prover provides the leaf l_i with a set of L "instructions."

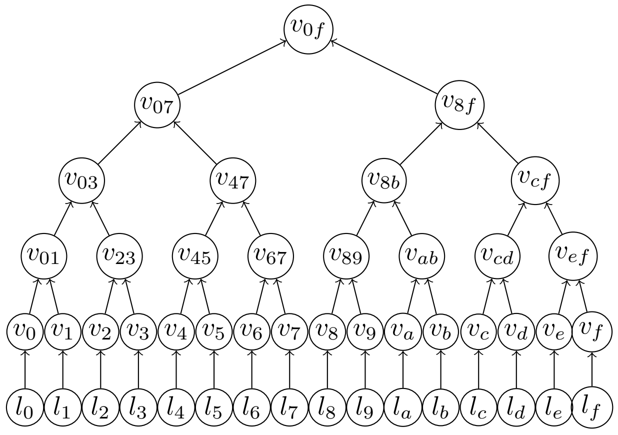
v_{0f}
v_{07}
v_{8f}
v_{03}
v_{47}
v_{8b}
v_{cf}
v_{01}
v_{23}
v_{45}
v_{67}
v_{89}
v_{ab}
v_{cd}
v_{ef}
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_a v_b v_c v_d v_e v_f
l_0 l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8 l_9 l_a l_b l_c l_d l_e l_f

Fig. 1. A Binary Merkle tree with 16 leaves.

For example, to prove that l_4 belongs to the tree, the prover provides the $L = 4$ instructions $(v_5, 0), (v_{67}, 0), (v_{03}, 1)$ and $(v_{8f}, 0)$. In each instruction, the first value is a hash and the second is an "order-bit." The verifier first computes $v_4 = h_l(l_4)$. Then following the instructions, the verifier computes $x = h(v_4 \parallel v_5)$, $x = h(x \parallel v_{67})$, $x = h(v_{03} \parallel x)$ and $x = h(x \parallel v_{8f})$. Note that in each instruction, the order-bit which informs the verifier how the result of the previous hash should be combined with first value in the instruction. The sequence of L hash operations performed based on the L instructions will result in an output that is the same as the root of the tree. As long as the hash function $h()$ is pre-image resistant, the verifier can now conclude that l_4 is indeed a part of the tree.

To modify the leaf l_4 to l'_4 the prover provides the verifier with i) the old leaf; ii) L instructions; iii) the new leaf; and iv) a justification for the modification. The verifier verifies i) that l_4 is part of the tree, and ii) the justification provided. If satisfied, to accept l'_4 as a replacement for l_4 the verifier first computes $v'_4 = h_l(l'_4)$ and uses the same set of L instructions (provided to map v_4 to the current root), to compute the new root.

A. Index Ordered Merkle Tree

In general, a leaf $l_j, 1 \leq j \leq N$ may contain a record with a record-index $i \in \mathcal{I}$ where $|\mathcal{I}| >> N$. For example, $\mathcal{I}$ could be the space of 128-bit IPv6 addresses or 48-bit MAC addresses, but N may be substantially smaller (for example, thousands).

If $N \geq |\mathcal{I}|$, each record-index can be associated with a fixed leaf-index $1 \cdots N$ in the tree. For example, record-index i may simply correspond to the i^{th}leaf. The verifier requiring the current information regarding record indexed i can then simply ask the prover to provide the i^{th}leaf (along with L instructions). If new information regarding entity i is received, leaf i can be updated. After this the prover will not be able to replay the old information regarding entity i.

However, in scenarios where $|\mathcal{I}| >> N$ only a small subset of values in $\mathcal{I}$ are actually associated with leaves $l_1 \cdots l_N$. If the verifier (who does not store the leaves) desires to know the current information regarding a record-index i, the prover should either provide the leaf which contains the information regarding i or, alternately, *prove that no such leaf exists*. Without this feature, the prover may be able to insert multiple leaves corresponding to index i, and thus be able to replay invalidated data regarding the index i.

In IOMT a leaf is a three-tuple of the form (i, θ, i'). This leaf corresponds to a record index i. The value θ could be a pre-image resistant hash of the entire contents of the record. The value i' is also an index and signifies that no leaf exists in the tree corresponding to an index that is *enclosed* by i and i'.

A value x is enclosed by (i, i') if $i < x < i'$, or if $x < i' < i$, or if $i' < i < x$. If $i = i'$ all values are enclosed - implying that i is the only leaf in the tree. The set of all current leaves are thus virtually ordered based on an index. where each index points to the next available index. The highest index wraps around and points to the lowest index. This approach is used in NSEC records in DNSSEC (domain name system security) for providing authenticated denial of queried records. An NSEC record of the form (`abc.example.com`, `abf.example.com`) proves that no record pertaining to a name `abe.example.com` exists. IOMT combines NSEC and Merkle hash trees.

The IOMT tree is initialized by providing the verifier with the root of an empty tree (all leaves set to $(0, 0, 0)$). The first inserted leaf points to itself as the next leaf, thereby proving that no other leaf exists. To add a leaf (C, θ_C) (or insert C in the place of an empty leaf $(0, 0, 0)$) the prover demonstrates that C does not currently exist in the tree. This is achieved by showing a valid leaf like (A, θ_A, E) which shows that no leaf exists for indices that fall between A and E. When C is inserted (in place of a currently empty leaf) the leaf for A will also need to be changed to (A, θ_A, C) to point to C, and the value (C, θ_C, E) replaces an empty leaf.

Similarly, to delete a leaf corresponding to an index S, the prover should provide two current leaves - the leaf to be deleted, say (S, θ_S, U) and a leaf (say (P, θ_P, S)) which points to S. The leaf for S needs to be set to $(0, 0, 0)$ and the leaf for P will now need to point to U, and thus modified to (P, θ_P, U).

As can be seen from the examples above, the process of either adding or deleting a leaf requires modifying two leaves simultaneously. Consider a scenario where two leaves (say l_1 and l_6) need to be modified concurrently. Note that the leaves l_1 and l_6 have a common parent (v_{07}) at height $n = 3$. To modify both leaves simultaneously, verifier is provided with

1) the old leaves l_1 and l_6;
2) $n - 1 = 2$ instructions $(v_0, 1)$ $(v_{23}, 0)$ to reach the left-child (v_{03}) of the common parent, starting from $v_1 = h(l_1)$;
3) $n - 1 = 2$ instructions $(v_7, 0)$ and $(v_{45}, 1)$ to reach the right-child v_{47} of the common parent, starting from $v_6 = h_l(l_6)$; the verifier can now compute the common parent $v_{07} = h(v_{03} \parallel v_{47})$.
4) $L - n = 1$ instruction(s) (in this case $(v_{8f}, 0)$) to reach the root from the common parent; and
5) the new leaves l'_1 and l'_6 (along with the justification for the changes).

After verifying that l_1 and l_6 are part of the tree, and the justification provided, the verifier can now compute a new root starting from values $v'_1 = h_l(l'_1)$ and $v'_6 = h_l(l'_6)$, using the same set of $2 \times (n - 1) + L - n$ instructions.

IV. TMMs and MANET TCB Functions

For our purposes, as we shall see in this section, the leaves of the IOMT are routing records indexed by the destination identity. Resource-limited trustworthy MANET modules (TMMs) store the root of the IOMT. The modules will also be capable of performing fixed sequence of hash and logical operations to accept RRs authenticated by TMMs of neighbors, update the database of RRs, and authenticate RRs for verification by TMMs of neighbors. By doing so, TMMs can ensure that untrusted nodes which store the RRs have no freedom to misrepresent or even hide routing information.

In general, every node announces its presence in a MANET subnet by sending a packet to its neighbors indicating a new sequence number, and perhaps time of expiry. This information traverses over multiple hops to other nodes (as second-hand, third-hand, ... , n^{th}hand information) in the subnet. Depending on the nature of the protocol, every node may receive and store one or more routing records (RR), originating from some or all nodes in the subnet.

Consider a routing record (RR) for a node D stored by a node X, of the form

$$r_D^A = [D \parallel q \parallel m \parallel \tau] \parallel A, \tag{1}$$

where q is a sequence number, m is a distance metric (say, hop-count to D), and τ is a time after which this RR should be discarded. This RR was received by X from a neighbor A, indicating a hop count $m-1$. The hop count was incremented by one before it was stored by X, indicating A as the next-hop in the path to D.

An RR for D indicating a metric $m = inf$ (where inf is a large number) indicates that D is unreachable. Such a situation can happen if, for example, after X received the RR from A, the node A had moved away from the neighborhood of X. An RR for D is updated by X if a new RR received by X has i) a higher sequence number, or ii) the same sequence number with a shorter metric.However, if the new RR is sent from the same neighbor that had sent the stored RR, the RR is updated even if it indicates the same sequence number and higher metric.

When X receives a query requesting a path to a node D, the node X with a valid RR for D is expected to provide its RR for D to the querier (and not forward the query onward). While the list of rules to be followed by every node is simple, the primary challenge is in ensuring that nodes will indeed adhere to such rules.

In the proposed approach every node possesses a trustworthy MANET module (TMM) which performs simple to ensure that i) nodes update their RRs consistent with some rules,

ii) nodes cannot advertise incorrect RRs, or even hide RRs. TMMs are deliberately constrained to be simple to ensure that they merit trust. The tasks performed by the TMM are in response to inputs provided to the TMMs over well defined interfaces. Such well-defined TCB functions performed by the TMM are

1) UpdateNeighborTable(),
2) AddDeleteLeaf(), and
3) RelayRoutingRecord(),

The function UpdateNeighborTable() receives time-stamped message authentication codes (MAC) (based on pairwise secrets) as inputs to maintain a table of active neighbors inside the TMM. The function AddDeleteLeaf() is used to add or delete leaves of an index-ordered Merkle hash tree (IOMT), where only the root of the tree is stored inside the TMM. The leaves of the tree are RRs, indexed by the destination. The function RelayRoutingRecord() is used to submit RRs from a neighbor (authenticated by a TMM), modify the RR subject to simple rules, and obtain MACs for updated RRs for verification at TMMs in neighboring nodes. The three functions which are assumed to be immutable, will serve as the TCB for a MANET nodes.

A. *Trustworthy MANET Modules (TMM)*

TMMs can be low complexity low cost chips, possibly similar to the trustworthy computing group (TCG) trusted platform modules (TPM). Like TPMs, TMMs will be non-programmable, and perform a fixed set of functions. It is assumed that that every TMM possesses

1) a unique identity (which is the same as the identity of the MANET node which houses the TMM);
2) an in-built cryptographic compression function $h()$ (for example, SHA-1);
3) a secret known only to the TMM; we shall denote such a secret in TMM X as ϕ_X;
4) some secrets provided by a trusted authority (like a key distribution center) which can be used to compute pairwise secrets with other nodes; thus two TMMs A and B can independently compute a pairwise secret K_{AB} using their respective secrets.
5) a clock-tick counter: a TMM X uses its clock-tick value t_x and a value t_{x_d} provided to the TMM (possibly during the boot-strapping process during which keys are provided to the TMM) to compute a value $t = t_x - t_{x_d}$. The value t computed by all TMMs are "sufficiently close," and related to the real time t (for example, number of microseconds elapsed since 0000 GMT Jan 1 2010).
6) limited (say 1 KB) volatile memory;

TMM functionality necessary for time synchronization and boot-strapping of TMMs are *not* discussed in this paper. For efficient approaches to facilitate pairwise keys (not discussed in this paper) between low complexity trustworthy modules see [12]- [14].

1) TMM Data Structures: Every TMM maintains a table of neighbors. In the neighbor-table each row consists of

1) the identity of the neighbor,
2) the time at which the neighbor was last heard from (the time at which an authenticated packet was received from the neighbor);
3) a pairwise key for the neighbor, and
4) the status (0,1, or 2) of the neighbor.

Nodes in the neighbor table with status 0 are not considered as neighbors. Status 1 indicates that bi-directionality of the link has *not* been confirmed; status 2 indicates a neighbor with a confirmed bidirectional link.

Apart form the neighbor table, the other values stored inside the TMM include some static protocol specific constants. Some examples of such parameters include

1) T_v: life-time of a newly created sequence number;

2) δ: permitted time difference between the times when a MAC was created by a TMM and the time a TMM in a neighbor verifies the MAC (accounting for processing delay and clock-drift).

3) δ_s time after which silent neighbors will be removed from the table (status will be set to 0).

4) inf a constant to denote infinite metric, and

5) r_0 the root of an empty IOMT;

In addition, TMMs maintain two dynamic values - the current sequence number q of the node, and the root of the IOMT. All static values, and the dynamic sequence number q, are assumed to be preserved even if the node is rebooted.[1] On resetting the TMM, the static parameters and the sequence number q are loaded and stored inside the TMM, with the root set to r_0; the neighbor table is empty.

2) Inputs and Outputs: The inputs to the three interfaces include values like leaves of the IOMT, RRs, MACs, time, and instructions to map a leaf to the root. The outputs of the TMM are MACs and the current time (according to the clock-tick counter inside the TMM).

An RR pertaining to a destination D, provided by a neighbor A is of the form

$$r_D^A = \mathbf{D} \parallel A \text{ where } \mathbf{D} = [D \parallel q \parallel m \parallel \tau]. \tag{2}$$

Associated with the r_D^A are values

$$h_D = h(\mathbf{D}), \text{ and } \theta_D = h(h_D \parallel A). \tag{3}$$

An IOMT leaf is of the form (D, θ_D, G) where D is the identity of the destination, θ_D (as in Eq (3)) is a one-way function of a RR for D, and G is the identity of the next node (in the increasing order of node identity) in the tree.

A leaf with $\theta = 0$, for example, $l_j = (D, 0, H)$, indicates an *uninitialized* RR for D. An initialized RR r_D^A with $\tau < t$ (where t is the current time) is an *expired* RR. An initialized RR is *invalid* if it has not expired, but has a metric $m = inf$ (which indicates a lack of a path to D).

[1]This could be achieved by authenticating the current state against a non-resettable counter inside the TMM. Specific strategies for this purpose are beyond the scope of this paper.

3) Message Authentication Codes: Consider an example where a TMM X with neighbors A, B and C in its neighbor table advertises an RR r_D^A (a route for D provided by a neighbor A) to its neighbors. The TMM X shares secrets K_{XA}, K_{XB} an K_{XC} respectively, with its neighbors. The pairwise secrets are also stored in the neighbor table. This broadcast by X to its neighbors is individually authenticated using message authentication codes (MAC) μ_A, μ_B and μ_C, which are computed as

$$\begin{aligned} \mu_A &= h(h_D \parallel t \parallel 1 \parallel K_{XA}) \\ \mu_B &= h(h_D \parallel t \parallel 0 \parallel K_{XB}) \\ \mu_C &= h(h_D \parallel t \parallel 0 \parallel K_{XC}) \end{aligned} \quad (4)$$

where h_D is computed as shown in Eq (3), and t is the current time (according to the clock inside the TMM X). Note that the MAC is computed differently for nodes like B and C (who can use RRs to update their RR database), and for A who should *not* use this RR to update its database (as this RR is an ACK for an RR from A). More specifically, that the MAC for A is computed using a flag 1 indicates that the MAC is an ACK.

For example, node B will be able to supply the RR $[D \parallel q \parallel m \parallel \tau]$ supplied by X along with the MAC μ_B to its TMM (TMM B) to demonstrate that

1) node X is a neighbor which sent some value h_D at time t (this information can be used to update the neighbor table of B); and
2) $[D \parallel q \parallel m \parallel \tau]$ is an authentic RR from X.

Node C can also use the value μ_C to update the neighbor table and demonstrate (to its TMM) the receipt of a new RR from X. However, node A can use the MAC μ_A (which is computed using a flag 1) only to update its neighbor table.

B. TCB Function UpdateNeighborTable()

A node A is a neighbor of X only if it has an entry in the neighbor table of X, with status 1 or 2. A node with status 1 or 2 will be added to the table only if a verifiable MAC is received from the node.

The inputs to UpdateNeighborTable() are i) an identity of a neighbor (say A); ii) a value V; iii) a MAC μ, iv) time t' at which the MAC was computed; and v) and a one bit flag b. The TMM X uses the pairwise secret K_{XA} to verify that

$$\mu = h(V \parallel t' \parallel b \parallel K_{XA}) \quad (5)$$

If the MAC is found to be authentic, and the difference between the time t' and current time t of the TMM X is less than a threshold δ, the neighbor table entry for A is updated. More specifically, i) if the last-heard field for A is currently $t_o < t'$, then the last-heard field is updated to t'; ii) if no entry exists currently for A, an entry is created, and the status set to 1. Furthermore, stale entries in the neighbor table - entries for which the difference between the last-heard time and current time is more than a threshold δ_s, are removed from the table.

If the MAC is authentic and $b = 1$, the implication is that the MAC is an acknowledgment. If the status of the neighbor A is currently 1 (bi-directionality not verified), receipt of an ACK is construed as proof of bi-directionality, and the status is set to 2.

If the MAC is authentic, $b = 0$, and[2] $V \neq t'$, the implication is that the value V is the hash of an RR provided by A. In this case the TMM outputs a "memorandum to itself" in the form of a "symmetric certificate" sc binding V to the provider A, and the current root r of the IOMT, as

$$sc = h(V \parallel A \parallel r \parallel \phi_X), \quad (6)$$

where ϕ_X is a secret known only to TMM X. The value sc can be submitted as proof to the TMM that a value V was sent by A till the time the root of the IOMT remains r. As we shall see soon, such certificates (or a "self-MAC") will serve as inputs to the RelayRoutingRecord() function to update the RR database.

An entry for a potential neighbor A can be initiated (status set to 0, and pairwise key K_{XA} computed and stored) without providing a valid MAC. In all three approaches [12] - [14] for facilitating pairwise keys between resource limited nodes, based on two inputs A and P_A (where A is the identity of a module and P_A is a non secret value associated with A), X can compute a pairwise secret K_{XA} by performing a small number of cryptographic hash operations. If the five inputs to UpdateNeighborTable() are $(A, P_A, 0, 0, 0)$, the TMM considers this as a request to add an entry for A in the neighbor table (with status set to 0), and computes the key K_{XA}.

While no node cannot be *added* as a neighbor (status 1 or 2) without a verifiable MAC, TMMs permit nodes to *remove* neighbors without offering any justification. A call UpdateNeighborTable$(Y, 0, 0, 0, 0)$ to the TMM removes the identified neighbor Y from its table (if an entry exists for Y). It is important to note that TMMs can only ensure that a node cannot perform active attacks like replaying old RRs or providing incorrect RRs, or hiding RRs. A TMM cannot however force a node to actually *send* the RR: after all, the communication interfaces are under the control of the node - not the TMM. The power granted to the untrusted node which houses the TMM to remove nodes from the TMMs "view of the neighborhood" is to provide the ability to deter selfish neighbors engaging in passive attacks like selective participation. A selfish node risks being ejected from the neighborhood of all nodes.

C. TCB Function AddDeleteLeaf()

The TCB function AddDeleteLeaf() is used for inserting and deleting leaves in an IOMT. Recall from Section III that two leaves have to be modified simultaneously to insert or delete a leaf. The inputs to the function AddDeleteLeaf() are thus

1) identity of the node to be deleted/inserted.
2) two leaves $l_i = (i_1, \theta_i, i_2)$ and $l_j = (j_1, \theta_j, j_2)$;
3) n - the height of the common parent of l_i and l_j
4) a sequence of $n - 1$ instructions $\mathbf{I}_i$ to be applied to $v_i = h_l(l_i)$ to obtain a value x;

[2]If $V = t'$, this is a "HELLO" message from A indicating only its current time.

5) a sequence of $n-1$ instructions $\mathbf{I}_j$ to be applied to $v_j = h_l(l_j)$ to obtain a value y;
6) L: height of the tree;
7) $L-n$ instructions $\mathbf{I}_r$ to map $h(x \parallel y)$ to the root;

The function AddDeleteLeaf() first verifies the current leaves against the root of the tree. If at least one leaf is empty, the TMM assumes that a leaf needs to be inserted.

Assume that destination identity A needs to be inserted to replace a leaf $l_i = (0,0,0)$. If the tree is empty (the current root is r_0), the TMM modifies the first leaf $l_i = (0,0,0)$ to $l_i' = (A,0,A)$. If not, the operations performed by the TMM (after verifying the leaves) are

1) verify that (j_1, j_2) encloses A (to ensure that no leaf for A exists currently);
2) create leaf $l_j' = (j_1, \theta_j, A)$ to replace l_j;
3) create leaf $l_i' = (A, 0, j_2)$ to replace l_i;
4) update root.

Now assume that F is indicated as the identity to be removed. In this case, $l_j = (j_1 = F, \theta_F, j_2)$ should be replaced with $l_j' = (0,0,0)$. Now the steps taken by the TMM are

1) verify $\theta_F = 0$ and $i_2 = F$;
2) create $l_j' = (0,0,0)$ and $l_i' = (i_1, \theta_i, j_2)$ to replace l_j and l_i;
3) update root.

The function AddDeleteLeaf() creates place-holders for destinations, with uninitialized RRs. Leaves are inserted with their θ values set to 0; and leaf can be deleted only if its θ value is zero. Uninitialized RRs can be initialized, modified (and initialized RRs can be uninitialized) using TCB function RelayRoutingRecord().

D. TCB Function RelayRoutingRecord()

Broadly, the purpose of the TCB function RelayRoutingRecord() in a TMM X is to create MACs (which can be sent to neighbors of X) corresponding to an RR which is stored and can be verified by X to be consistent with the root of it's tree. The MACs are created as explained in Section IV-A3 (for authenticating an RR received from a neighbor A, the MAC for A includes a flag set to 1 and for all other neighbors the flag is set to 0).

In general, inputs to RelayRoutingRecord() are i) a currently stored RR along with instructions to map the RR to the current root r; and ii) a fresh RR received from a neighbor, along with a symmetric certificate (to prove that the hash of the RR was received from the neighbor when the root was r). Recall that the symmetric certificate was created as an output of UpdateNeighborTable(). The function RelayRoutingRecord() updates the RR subject to some simple rules, and advertises the updated RR to all neighbors (by creating MACs).

In some scenarios, the new RR may not cause the old RR to be updated. In some scenarios the function RelayRoutingRecord() may be called *without* a new RR, to simply advertise a current RR, or advertise a current RR after modifying it. A current RR r_D^A may need to be modified without an external trigger if: i) neighbor A which provided the RR is no longer a neighbor, ii) time of validity of the RR has expired, or iii) the leaf for the corresponding RR is uninitiated.

Assume that the RR pertaining to a node S currently stored in the RR database of TMM X is $r_S^A = \mathbf{S} \parallel A$, where $\mathbf{S} = [S, q, m, \tau]$, and that the corresponding leaf is $l_s = (S, \theta_S, V)$ (where $\theta_S = h(h_S \parallel A)$, where $h_S = h(\mathbf{S})$). Also assume that new information $\mathbf{S}' = [S, q', m', \tau']$ regarding S has been received from a neighbor id. The inputs to the interface RelayRoutingRecord() to process the new RR are

1) current RR $\mathbf{S} \parallel A$
2) current leaf $l_s = (S, \theta_S, V)$;
3) L instructions to map l_s to the current root r.
4) new RR $\mathbf{S}' \parallel id$
5) a symmetric certificate sc $sc = h(h_S \parallel id \parallel r \parallel \phi_X)$ (where ϕ_X is a secret known only to TMM X) to authenticate the new RR received from neighbor id

The TMM verifies $\mathbf{S} \parallel A$ is consistent with the value θ_S in l_s, and that the L instructions map the leaf l_s to the current root r. The TMM then verifies the new RR $\mathbf{S}' \parallel id$ using sc.

If consistent, the RR can be updated subject to some rules. For example, RR will be updated if

1) $q' > q$: fresher sequence number;
2) $q' = q$ and $m' < m-1$: same sequence number, shorter metric;
3) $q = q'$, $id = A$, $m' \neq m-1$: same sequence number, different metric, provided by the same neighbor who provided the currently stored RR.
4) $\theta_S = 0$: the current RR for S is uninitialized.

As explained in Section IV-A3, a MAC is computed for all neighbors (with status 1 or 2).

1) $sc = 0$: Unauthenticated Requests: If $sc = 0$, the TMM interprets this as a request to relay (and possibly update) a record without an authenticated input from another TMM. The TMM first checks validity of the current RR and updates the RR as follows

1) if A is not an active neighbor and $t < \tau$ (where t is the current time according to the TMMs clock), the TMM sets $m = inf$, and updates θ_S, and the root r.
2) if $t \geq \tau$, the TMM sets $\theta_S = 0$ and updates root r;
3) if $\theta_S = 0$ the TMM sets $\theta_S = h(h(S \parallel q = 0 \parallel inf \parallel 0) \parallel 0)$ and updates the root r (uninitialized RR is initialized to an invalid,expired RR).

Irrespective of whether the RR was updated, the now-current RR is authenticated to its neighbors.

If $sc = 0$ and $S = X$ (the TMMs own identity) the TMM interprets this as a request to announce its height (0) with a fresh sequence number. In this case the TMM X increments its own sequence number to q', computes $h_X = h(X \parallel q' \parallel 0 \parallel t')$ where $t' = t + T_v$, and T_v is a static parameter which specifies the duration of validity information corresponding to a sequence number), and outputs a MAC for each neighbor.

If all input fields to a call RelayRoutingRecord() are empty the TMM outputs a MAC for every node in the neighbor table - even ones with status set to 0. For example, for to a node Z

in the table with pair-wise key K_{XZ} the MAC is computed as

$$\mu_Z = h(t \parallel t \parallel 0 \parallel K_{XZ}). \quad (7)$$

E. Using the TMM Interfaces

We shall now provide a broad overview of how the three TMM interfaces can be used for securing a protocol like AODV. Every packet transmitted by a node confirms to the plain AODV protocol - the only difference is that every packet includes a set of MACs - one for each neighbor, and the time t at which the MAC was computed.

Consider a node X (with TMM X) which is powered on. By overhearing packets, the node X recognizes that it has nodes (say) A, B, C in its neighborhood. At this point, as A, B, C do not consider X as their neighbor, the packets overheard by X cannot be cryptographically verified by the TMM X (thus, while node X can "sense" its neighbors, the TMM of X cannot).

X uses the UpdateNeighborTable() interface to add A, B and C to the neighbor table, with status set to 0. Now X uses its RelayRoutingRecord() interface (with all fields set to 0) to obtain a time-stamped MAC for each potential neighbor. Such MACs from X can be used by A, B and C to add X to their neighbor tables (with status set to 1). Subsequent packets from A, B and C will include a MAC for X which can be used by X to add them to its neighbor table. Whenever an RR sent by a node X is ACKed by a neighbor A, X will mark A's neighbor status as 2 (bi-directional link verified).

Assume that a node S in the subnet desires to send a route request for a node D. A route request from S to D provides i) a valid RR regarding the initiator S, and ii) an invalid RR ($m = inf$) corresponding to a destination D. The initiator makes two calls to RelayRoutingRecord() - to create an updated RR for itself (with a new sequence number), and to get MACs for an invalid RR for D (which the node cannot do if it has a valid RR for D). Individual MACs are created for both RRs (one for each neighbor). The nodes which receive the RREQ supply a valid RR for S to update their RR database. Following this they supply the invalid RR for D. If a node has a valid RR for D the node cannot relay the RREQ - as the TMM will only create a MAC for a valid RR for D.

To send a route error (RERR) message for a destination D, a node is required to send an invalid RR for D to its neighbors. The TMM does not know, or care, if an invalid RR is used for an RREQ, or a route error message. All that the TMMs know are a simple set of rules to update a leaf (and consequently, the root) and creating MACs.

If a node processing or creating an RREQ for a destination D does not currently have a leaf for D, an uninitialized entry for D is created using AddDeleteLeaf(). The RREQ creator can initialize the RR (to infinite metric, last known sequence number 0, and time of expiry 0) by using the interface RelayRoutingRecord(). Intermediate nodes can initialize their RR for D using the received RR.

V. Conclusions

Securing MANET routing protocols mandates ensuring that nodes do not advertise incorrect routing information, or hide routing information. Conventional approaches for securing MANETs rely on a variety of techniques including carrying over of cryptographic authentication such that a node can verify that an RR provided by a node Y is consistent with the same RR provided to Y by a neighbor of Y. Carrying over authentication, does not prevent attacks by colluding nodes, does not inhibit the ability of a node to hide information, and demands substantial overhead. Due to the substantial complexity and overhead associated with mechanisms for carrying over authentication, secure extensions of MANET routing protocols dissuade many possible optimizations; while such optimizations can make the protocols more efficient, they also render the problem of *authenticating* RRs substantially more complex.

The need for trusted boundaries in which co-operative routing tasks are carried out has been addressed by some researchers. In [16] the authors include the wireless transceiver inside the trust boundary. In [17] the trusted computing module has complex features built into the wireless driver (executed within the confines of the trusted module). In [18] explicit consideration is given to the need for lowering the complexity of tasks to be performed inside the trusted boundary. The scheme employs "nuglets of currency" protected by smart-cards to promote faithful forwarding of packets. More recently Gaines et al [19] have proposed a generic dual-agent approach to MANETs where some desired characteristics of a trustworthy network agent (like low computational and storage requirements) are enumerated.

The motivation for this paper stems from the very essence of the notion of trustworthy computing - that simple tasks performed inside a trustworthy boundary can serve as a trusted computing base (TCB), which can be *amplified* to provide complex assurances. In this paper we outlined simple TCB functions for MANET nodes. In designing the TCB functions we have placed some common-sense restrictions like restricting the operations to simple, hard-wired sequences of logical and cryptographic hash operations. Such TMMs with simple functionality can be realized at low cost. Due to their simplicity their hard-wired functionality can be verified easily.

One of the primary challenges in the design of the TCB functions was a mechanism for maintaining the integrity of the dynamic database of RRs stored outside the TMM. If the label corresponding to each leaf is fixed, (for example, leaves 1 to N contain information regarding entities 1 to N respectively) simple Merkle trees can achieve this goal. However, in scenarios where one-to-one correspondence between identities and leaves are not possible, the index-ordered Merkle tree (IOMT) which combines the concept of Merkle trees with the NSEC approach used in DNSSEC, is necessary.

As a first attempt in deriving tangible TCB functions for MANET nodes, we have focused protocols where only one RR pertaining to a destination is stored in every node. However, we believe that simple extensions and generalizations to the TCB function RelayRoutingRecord() can result in a TCB function suitable for different types of routing protocols. Research on generalization of the function RelayRoutingRecord() is underway.

References

[1] B. Lampson, M. Abadi, M. Burrows, E. Wobber, "Authentication in Distributed Systems: Theory and Practice," ACM Transactions on Computer Systems, 1992.

[2] Y-C Hu ,A Perrig,. D B.Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," Journal of Wireless Networks, **11**, pp 11–28, 2005.

[3] S. Weiler, J. Ihren, "RFC 4470: Minimally Covering NSEC Records and DNSSEC On-line Signing," April 2006.

[4] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose "RFC 4033: DNS Security Introduction and Requirements," March 2005.

[5] C Perkins, P Bhagvat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," ACM SIGCOMM Symposium on Communication, Architectures and Applications, 1994.

[6] C. Perkins, E.Royer, S. Das "Ad hoc On-demand Distance Vector (AODV) Routing, Internet Draft, draft-ietf-manet-aodv-11.txt, Aug 2002. The 6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2002), 2002.

[7] D. Johnson, D. Maltz, Y-C. Hu, J. Jetcheva, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks," Internet Draft, draft-ietf-manet-dsr-05.txt, June 2001.

[8] V. D. Park, M. S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," pp 1405–1413, INFOCOM 1997.

[9] J. Kim, G. Tsudik, "SRDP: Securing Route Discovery in DSR," IEEE Mobiquitous'05, July 2005.

[10] M.G.Zapata, N.Asokan, "Securing Ad hoc routing protocols," WISE-02, Atlanta, Georgia, 2002.

[11] R.C. Merkle "Protocols for Public Key Cryptosystems," In Proceedings of the 1980 IEEE Symposium on Security and Privacy, 1980.

[12] M. Ramkumar, "The Subset Keys and Identity Tickets (SKIT) Key Distribution Scheme," IEEE Transactions on Information Forensics and Security, **5**(1), pp 39–51, March 2010.

[13] M. Ramkumar, "On the scalability of a "non-scalable" key distribution scheme," IEEE SPAWN, Newport Beach, CA, June 2008.

[14] M. Ramkumar, "Trustworthy Computing Under Resource Constraints With the DOWN Policy," IEEE Transactions on Secure and Dependable Computing, March 2008.

[15] TCG Specification: Architecture Overview, Specification Revision 1.4, 2nd August 2007.

[16] J-H. Song, V. Wong, V. Leung, "Secure Routing with Tamper Resistant Module for Mobile Ad Hoc Networks," ACM SIGMOBILE Mobile Computing and Communications Review, vol. 7, no. 3, ACM Press, New York, Jul. 2003.

[17] M. Jarrett and P. Ward, "Trusted Computing for Protecting Ad-hoc Routing," Proceedings of the 4th Annual Communication Networks and Services Research Conference, IEEE Computer Society, May 2006.

[18] J-P. Hubaux, L Buttyan, S. Capkun, "Quest for Security in Mobile Ad Hoc Networks," Proceedings of the ACM MOBIHOC 2001.

[19] B. Gaines, M. Ramkumar, "A Framework for Dual Agent Routing Protocols for MANETs," IEEE Globecom 2008, New Orleans, LA, Nov 2008.