Institute of Advanced Engineering and Science
International Journal of Information & Network Security (IJINS))
Vol. 6, No. 6, Deember 2014, pp. 392 – 409
ISSN: 2089-3299

Ordered Merkle Tree: A Dynamic Authenticated Data Structure for Security Kernels

Somya D. Mohanty* **and Mahalingam Ramkumar****
*Social Science Research Center, Mississippi State University
**Computer Science and Engineering, Mississippi State University

Article Info

Article history:
Received
Revised
Accepted

Keyword:
Security Kernels
Data Structures
Third keyword
Network Protocols
Border Gateway Protocol (BGP)
Authenticated Data Structures (ADS)

Corresponding Author:
Somya D. Mohanty
Social Science Research Center,
Mississippi State University,
1 Research Blvd., Suite #103,
Starkville, MS 39759
662-325-3356
mohanty.somya@uncg.edu

ABSTRACT

We introduce a family of authenticated data structures — Ordered Merkle Trees (OMT) — and illustrate their utility in security kernels for a wide variety of sub-systems. Specifically the utility of two types of OMTs: a) the index ordered merkle tree (IOMT) and b) the range ordered merkle tree (ROMT), are investigated for their suitability in security kernels for various sub-systems of Border Gateway Protocol (BGP), the Internet's inter-autonomous system routing infrastructure. We outline simple generic security kernel functions to maintain OMTs, and sub-system specific security kernel functionality for BGP sub-systems (like registries, autonomous system owners, and BGP speakers/routers), that take advantage of OMTs.

1. INTRODUCTION

Any system can be seen as a network of sub-systems, each with a specific role in the operation of the system, interacting with each other according to system-specific and/or role-specific rules. For an ever increasing range of systems, some or all sub-systems take the form of a computer, or a collection of computers (most often a server with one or more back-end servers). For example,

1) sub-systems in the domain name system (DNS) have roles like *zone authorities*, who create DNS resource records (RR) pertaining to the zone; *authoritative name servers*, that are chosen by the zone authority to disseminate DNS RRs for the zone; and *local (or preferred) name servers*, that iteratively query authoritative name servers to resolve queries from clients.

2) sub-systems in the inter-domain routing infrastructure for the Internet — the Border Gateway Protocol (BGP) — have different roles like *autonomous system (AS) owner*; *AS registry*, that assigns AS numbers to AS owners; *IP registry* that issues (through IP registrars and ISPs) chunks of IP addresses, or IP prefixes[1] to AS owners; and *BGP speakers* for an AS, authorized by the AS owner to originate routes for IP prefixes owned by AS.

Undesired functionality in any hardware/software component of a sub-system may be exploited by an attacker to cause a sub-system to misbehave. Undesired functionality may be deliberately hidden malicious functionality (HMF), or accidental bugs. Attackers who exploit undesired functionality may be personnel with legitimate access to the sub-system, or any one who can take advantage of remotely exploitable HMF/bug to exert some control over

[1] An IP prefix is a chunk of 2^n consecutive addresses, where $32 - n$ MSBs of all addresses in the chunk are the preserved.

the sub-system. For example, an attacker can a) compromise a BGP speaker (in a router) to send incorrect routing information; or b) compromise a computer used by the AS administrator to modify the AS policies/preferences; or c) compromise a computer of an administrator in the IP / AS / DNS registry to make duplicate address / AS number assignments.

1.1. Security Kernel

It is far from practical to assure the integrity of *every* hardware/software component in *every* component of *every* sub-system. One possible approach to secure systems is to mandate that all important sub-systems should be associated with an appropriate *security kernel* that vouches for the integrity of (system-specific and role-specific) tasks performed by the sub-system. Specifically, all components of the sub-system are assumed to be untrustworthy; only the security kernel is trusted.

The security kernel for a system/sub-system is also referred to as the *trusted computing base* (TCB) for the system/sub-system. The TCB for any system is "a small amount of software and hardware that security depends on, and that we distinguish from a much larger amount that can misbehave without affecting security" [1]. For purposes of this paper, the exact nature TCB is not important. For example, the TCB for any sub-system could take the form of a dedicated hardware security module, or a software module executed on a general purpose platform, with some special protections [2] to guarantee that the security kernel will run unmolested, etc.

In the rest of this paper we shall assume that the security kernel for a sub-system is a set of functions executed by a read-proof and write-proof module $\mathbf{T}$. That the module is read-proof implies that secrets protected by the module (for authentication of the module) can not be exposed; that the module is write proof implies that the designed security kernel functionality of the module can not be modified. It is however essential that the security kernel functionality is *deliberately* constrained to be simple — to permit consummate verification of the functionality, and thereby, rule out the presence of undesired functionality *within* the security kernel.

Some of the components of the security kernel will necessarily be specific to the nature of the sub-system whose operation is assured by the module — the security kernel functionality for a DNS server will be different from that of an IP registry or a BGP speaker. Nevertheless, to simplify testing of the security kernel functionality, it is advantageous to possess efficient *re-usable* components of the security kernels, with potential to be useful in a wide range of sub-systems. The specific contributions of this paper are

1. an efficient reusable authenticated data structure (ADS), an ordered merkle tree (OMT), and
2. illustration of the utility of OMTs in a broad range of security kernels (for a broad range of sub-systems).

1.2. Ordered Merkle Tree

An ADS [3, 4, 5, 6, 7] is a strategy for obtaining a concise cryptographic commitment for a set of records. Often, the commitment is the root of a hash tree. Any record can be verified against the commitment by performing a small number of hash operations. An ordered merkle tree (OMT) is an ADS that is derived as an extension of the better known merkle hash tree. Similar to a plain merkle tree, an OMT permits a resource (computation and storage) limited module to track the records in a dynamic database of *any* size, maintained by untrusted components of the associated sub-system. Using an OMT (instead of a plain merkle tree) permits the resource limited module to additionally infer a few other "useful holistic properties" regarding the database.

For illustrating the broad utility of OMTs, we explore the security kernel functionality necessary for assuring the operation of various BGP sub-systems like IP and autonomous system (AS) registry/registrars, AS owners, and BGP speakers, etc.

The rest of this paper is organized as follows. In Section 2. we introduce OMTs, and discuss two types of OMTs — the index ordered merkle tree (IOMT) and the range ordered merkle tree (ROMT). In Section 3. we provide an overview of BGP. We enumerate the desired assurances regarding the operation of BGP and suggest high level designs of the security kernel functionality utilizing OMTs to guarantee the desired assurances (to the extent the security kernels are trusted). In Section 5. we suggest other possible applications of OMTs and offer our conclusions.

2. ORDERED MERKLE TREE

The merkle hash tree [8] is a data structure constructed using repeated applications of a a pre-image resistant hash function $h()$ (for example, SHA-1). Figure 1 depicts a tree with $N = 16$ leaves. In practical merkle tree applications each leaf can be seen as a record belonging to some database.

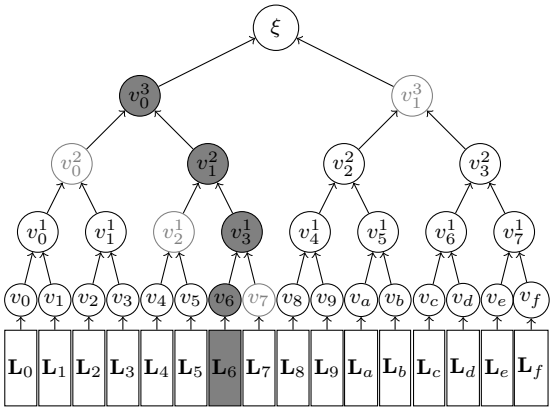


Figure 1. A binary hash tree with 16 leaves. Nodes v_6, v_3^1, v_1^2, v_0^3 (filled gray) and root ξ are ancestors of leaf $\mathbf{L}_6$. The set of "siblings of ancestors," viz., $\mathbf{v}_6 = \{v_7, v_2^1, v_0^2, v_1^3\}$ are "complementary" to v_6. The root is a commitment to all leaves. The complementary nodes $\mathbf{v}_6 = \{v_7, v_2^1, v_0^2, v_1^3\}$ are (together) commitments to all leaves *except* $\mathbf{L}_6$.

A tree with N leaves has a height of $L = \log_2 N$. At level 0 of the tree are N leaf-nodes, one corresponding to each leaf, typically derived by hashing the leaf. At the next level (level 1) are $N/2 = N/2^1$ nodes, each computed by hashing together a pair of "sibling" nodes in level 0. Level i has $N/2^i$ nodes computed by hashing a pair of siblings in level $i-1$, and so on, till we end up with a lone node ξ at level L — the root of the binary tree. A tree with $N = 2^L$ nodes has $2N-1$ nodes distributed over $L+1$ levels, where $L = \lceil \log_2 N \rceil$.

Two nodes node v_i^j and v_{i+i}^j at level j are siblings if i is even (else v_{i-1}^j and v_i^j are siblings). Two siblings — the left sibling u and the right sibling v are hashed together to obtain the parent node as $p = h(u, v)$. Any node in a binary tree can be seen as a root of a sub-tree, or a leaf-node of a subtree. Corresponding to a leaf node v in a sub-tree of height k with root y is a set k *complementary* nodes $\mathbf{v}$ that map the leaf node v to sub-tree root y. The complementary nodes of v include a) the sibling of v, and b) the siblings of all ancestors of v. The complementary nodes of a node v can collectively be seen as the set of commitments to all *other* nodes (except v) in the sub-tree.

In a sub-tree of height k, let $u_0^0 \cdots u_{2^k-1}^0$ represent the 2^k nodes at the lowest level (level 0). The immediate parent of a node u_i^0 is $u_{i_1}^1$ at level one, where $i_1 = i/2$ (if i is even) or $(i-1)/2$ (if i is odd). In this fashion, one can readily determine the indexes $u_{i_1}^1 \cdots u_{i_{k-1}}^{k-1}$ of all ancestors of u_i^0, and thereby, the siblings of the ancestors (the sibling of v_m^n is v_{m+1}^n if m is even, or v_{m-1}^n is m is odd).

Given the value v_i^0, the index i of the leaf node, and the set of k complementary nodes, it is trivial to identify the sequence of k hash operations necessary to map a leaf node to the root. We shall represent by

$$y = f_m(v_i, i, \mathbf{v}_i), \tag{1}$$

a sequence of k hash operations to obtain the sub-tree root y from a leaf-node with value v and position index i.

2.1. Verification and Update Protocols

Protocols that use a binary hash tree can be seen as a two-party protocol involving a prover and a verifier. For the purposes of this paper, the prover is an untrusted sub-system, and the verifier is a trusted module $\mathbf{T}$. The prover stores all N records *and* $2N-1$ nodes (cryptographic hashes) of the tree. The verifier $\mathbf{T}$ stores only the root of the tree.

Corresponding to any leaf $\mathbf{L}$, the prover (who maintains the entire tree) can readily identify the set of complementary nodes $\mathbf{v}$. If the prover supplies the leaf $\mathbf{L}$ along with the complementary hashes, the verifier can readily compute $f_m(v = h(\mathbf{L}), i, \mathbf{v})$. If the root stored by the verifier r is the same as $f_m(v, i, \mathbf{v})$, the verifier is simultaneously convinced of a) the integrity of the leaf $\mathbf{L}$, *and* b) the integrity of the complementary hashes $\mathbf{v}$.

Note that by construction of the tree it is guaranteed that $r = f_m(\mathbf{L}, i, \mathbf{v})$. That the hash function is pre-image resistant guarantees that the prover cannot determine $\mathbf{L}' \neq \mathbf{L}$ or $\mathbf{v}' \neq \mathbf{v}$ that satisfies $r = f_m(\mathbf{L}', i, \mathbf{v})$ or $r = f_m(\mathbf{L}, i, \mathbf{v}')$ (for any i).

Having demonstrated the integrity of the stored record to the module $\mathbf{T}$, if the prover can also demonstrate a legitimate need to modify the record $\mathbf{L}$ to $\mathbf{L}'$, the verifier simply computes $v' = h(\mathbf{L}')$ and updates the root to $r' = f_m(v', i, \mathbf{v})$. Once the leaf has been modified, the old leaf can no longer proved to be consistent with the new

root. However, all other leaves will still remain consistent with the new root (as the complementary nodes $\mathbf{v}$ are the commitments for *all other* leaves).

Note that verifying the integrity of any record in the database, or updating a record will require the security kernel to perform merely $\log_2 N$ hash function evaluations, where N is the number of leaves in the tree.

2.2. OMT Leaves and Nodes

An ordered merkle tree (OMT) is an extension of the merkle tree with the imposition of a special structure for the leaves of the tree. Every leaf is of the form.

$$\mathbf{L} = (A, A', \omega_A) \tag{2}$$

Corresponding to a leaf (A, A', ω_A) is a leaf node computed as

$$\begin{aligned} v_A &= H_L(A, A', \omega_A) \\ &= \begin{cases} 0 & A = 0, \\ h(A, A', \omega_A) & A \neq 0. \end{cases} \end{aligned} \tag{3}$$

In addition, unlike a plain merkle tree which is intended primarily for dynamic databases with a static number of records (leaves), OMTs are intended to be used for scenarios where leaves may need to be inserted/deleted. For this purpose it is advantageous to redefine the operation of mapping two siblings u and v to their parent p as

$$p = H_V(u, v) = \begin{cases} u & \text{if } v = 0 \\ v & \text{if } u = 0 \\ h(u, v) & \text{if } u \neq 0, v \neq 0 \end{cases} \tag{4}$$

In other words, the parent of two nodes is the hash of the two child nodes *only if both* children are non-zero. If any child is zero, the parent is the same as the other child. The parent of $u = v = 0$ is $p = 0$.

An OMT leaf with the first field set to zero is an *empty* leaf, represented as Φ. The leaf hash corresponding to an empty leaf is 0. As introducing an empty leaf node (corresponding to an empty leaf) does not affect any other node of the tree, any number of empty leaves may be seen part of the tree.

2.3. OMT Types

OMTs can be seen as falling under two broad categories depending on the interpretation of the first two values. In the first category are index ordered MTs (IOMT), where the first value is interpreted as an index, the second value is the next higher index in the tree. For the leaf corresponding to the highest index the next index is the least index. The third value ω_A in a leaf (A, A', ω_A) provides some information regarding index A. For example, ω_A could be the hash of the contents of a database record with index A. It is also possible that ω_A is a root of another OMT, in which case A is an index of a database (which may consist of any number of indexed records).

In an IOMT, existence of a leaf like $(432, 562, \omega)$ indicates that no leaf exists for indexes between 432 and 562. A wrapped around leaf like $(796, 241, \omega)$ indicates that no leaf exists for indexes greater than 796, *and* for indexes less that 241.

For range ordered MT (ROMT) the values A and A' represent the range $[A, A')$ of some quantity associated with the third value ω_A. For example, a leaf like $(432, 562, \omega)$ indicates that the quantity ω is associated with a range $[432, 562)$ (or $432 \leq x < 562$). For example, an ROMT may be used to represent a *look up table* (LUT) for some function $y = f(x)$. In such an ROMT each leaf indicates a range of the independent variable x, corresponding to which the function evaluates to the dependent variable $y = \omega$ (the third value in the leaf).

2.4. OMT Properties

Some of the important properties of OMTs are as follows:

1. The leaf hash corresponding to an empty leaf Φ is zero.
2. A tree with root 0 can be seen as a tree with *any* number of empty leaves.
3. For a tree with a single leaf, the leaf hash is the same as the root of the tree.
4. The existence of a leaf (A, A, ω_A) in an OMT indicates that the leaf is the only leaf in the tree (in which case the root will be the same as the leaf hash $H_L(A, A, \omega_A)$).

5. Existence of a leaf like $(1, 3, \omega_1)$ is proof that no leaf exists with first field in-between 1 and 3. Existence of a leaf like $(7, 1, \omega_7)$ is proof that no leaf exists with first field less than 1 *and* that no leaf exists with first field greater than 7.

6. As leaves are ordered virtually, the actual physical ordering of leaves has no inherent meaning. Thus, swapping leaves of an OMT does not affect the integrity of the database represented by the OMT. For example, both

$$\begin{array}{l}(1,3,\omega_1),(3,4,\omega_3),(4,7,\omega_4),(7,1,\omega_7) \text{ and} \\ (3,4,\omega_3),(1,3,\omega_1),(4,7,\omega_4),(7,1,\omega_7)\end{array} \tag{5}$$

represent an identical database with four records — either an IOMT corresponding to four indexes 1,3,4, and 7, or an ROMT for four ranges $[1, 3)$, $[3, 4)$, $[4, 7)$ and $\{7, 1\}$ representing all values greater than or equal to seven, or less than 1).

7. For both IOMT and ROMT, a leaf with a first field A can be inserted only if a leaf with first two fields that *circularly encloses* A exists. (B, B') is a circular enclosure for A only if

$$(B < A < B') \text{ OR } (B' < B < A) \text{ OR } (A < B' < B) \tag{6}$$

For inserting a leaf the contents of two leaves in the tree will need to be modified; and empty leaf Φ will be modified to become the newly inserted leaf with first values as (A, B'), and the second value of the enclosing leaf will be modified from (B, B') to (B, A).

8. A place-holder is a non-empty leaf whose insertion does not change the interpretation of the database. For an IOMT, a leaf of the form $(A, A', 0)$ (third value zero) is a place holder. Introduction of a place holder for an index A does not change the database in any way, as both existence of place holder for index A and non-existence of a leaf for index A implies that "no record exists for index A." Thus,

$$\begin{array}{l}(3,4,\omega_3),(1,3,\omega_1),(4,7,\omega_4),(7,1,\omega_7) \text{ and} \\ (3,4,\omega_3),(1,3,\omega_1),(4,5,\omega_4),(5,7,0),(7,1,\omega_7)\end{array} \tag{7}$$

which correspond to before and after insertion of a place holder for an index 5, represent an identical database.

9. For an ROMT, a place holder is a leaf with third value the same as the third value of the enclosing leaf. Specifically, inserting a leaf can be seen as a process of splitting a leaf (for example), $(4, 7, \omega_4)$ into two leaves (for example) $(4, 5, \omega_4)$ and $(5, 7, \omega_4)$. Specifically, both

$$\begin{array}{l}(1,3,a),(3,4,b),(4,7,c),(7,1,d) \text{ and} \\ (1,3,a),(3,4,b),(4,5,c),(5,7,c),(7,1,d)\end{array} \tag{8}$$

represent an identical database. Before insertion, the leaf $(4, 7, c)$ indicated that values $4 \leq x < 7$ are associated with c. Nothing has changed after the range is split into two, as values $(4 \leq x < 5)$ *and* values $(5 \leq x < 7)$ are associated with the same quantity c.

10. While operations like swapping leaves in any OMT or insertion/deletion of a place holder do not change the contents of the database, they will result in a change in the root of the tree — say from r to r'. Such roots are considered as *equivalent* roots.

2.5. OMT Functions for Security Kernels

The module $\mathbf{T}$ is assumed to possess limited protected storage, and expose well defined interfaces to the associated untrusted sub-system. Such interfaces can be used by an untrusted sub-system (say) A to demonstrate the integrity of databases stored by the sub-system, and request $\mathbf{T}_A$ associated with sub-system A to attest verified records.

For attesting records or contents of records (for verification by other sub-systems, or security kernels in other sub-systems) every module is assumed to possess a unique identity, and secrets used for authenticating messages. For example, the secret could be a private component of an asymmetric key pair, which is used for signing messages. In this case, the public key of the module is certified by a trusted key distribution center, attesting the integrity of the module. Alternately one or more secrets could be provided by a trusted key distribution center to each module. Only modules that have been verified for integrity and issued such secrets by the trusted key distribution centers will be able to use their secrets to compute a pairwise secret with other modules attested by the KDCs. Such pairwise secrets may be used to compute message authentication codes for attesting the integrity of the contents of a record.

Apart from secrets provided by trusted KDCs or certified by trusted certificate authorities, every module is assumed to spontaneously generate a random self-secret χ which is used for authenticating memoranda to itself. For

$F_{bt}(x, x', i, \mathbf{v}_x)$ {
$y \leftarrow f_m(x, i, \mathbf{v}_x); y' \leftarrow (x = x')? y : f_m(x', i, \mathbf{v}_x);$
RETURN $h(U1, x, y, x', y', \chi);$
}
$F_{cat1}(x_1, y, x'_1, y', \rho_1, x_2, z, x'_2, z', \rho_2)$ {
IF $(\rho_1 \neq h(U1, x_1, y, x'_1, y', \chi))$ RETURN;
IF $((\rho_2 = 0) \wedge (x_1 = x'_1))$ RETURN $h(V1, x_1, y, \chi);$
IF $(\rho_2 \neq h(U1, x_2, y_2, x'_2, y'_2, \chi))$ RETURN;
IF $((y = x_2) \wedge (y' = x'_2))$ RETURN $h(U1, x_1, z, x'_1, z', \chi);$
$p \leftarrow H_V(y, z); p' \leftarrow H_V(y', z');$
RETURN $h(U2, x_1, x_2, p, x'_1, x'_2, p', \chi);$
}
$F_{cat2}(x_1, x_2, y, x'_1, x'_2, y', \rho_1, z, z', \rho_2)$ {
IF $(\rho_1 \neq h(U2, x_1, x_2, y, x'_1, x'_2, y', \chi))$ RETURN;
IF $((\rho_2 = 0) \wedge (x_1 = x'_1) \wedge (x_2 = x'_2))$ RETURN $h(V2, x_1, x_2, y, \chi);$
IF $(\rho_2 \neq h(U1, y, z, y', z', \chi))$ RETURN;
RETURN $h(U2, x_1, x_2, z, x'_1, x'_2, z', \chi);$
}

Figure 2. Verification and Update Memoranda.

example, after executing (say) $z = f_m(x, i, \mathbf{v})$ a module may issue a memoranda to itself to remind itself that it has already verified that "z is an ancestor of x." As we shall see very soon, the self-memoranda in this scenario is a value $\rho = h(V1, x, z, \chi)$ computed as a function of the type $V1$ of the memoranda, the values x and z, and the secret χ. No entity other than the module can fake such a memorandum. Thus, if values x, z, and ρ are provided as inputs to the module, the module can safely conclude that "z is an ancestor of x."

In the rest of this section we provide an algorithmic description of generic OMT functions suitable for security kernels for a wide range of systems/ sub-systems. OMT functions issue different types of self-memoranda. Such self-memoranda may then be used by other system-specific (or role-specific) security kernel components of the same module. As an illustration of how such memoranda can be used by other system-specific security kernel components of the same module, in a later section we outline the use of such memoranda in security kernels for various BGP sub-systems.

2.6. OMT Memoranda

Five different types of memoranda are issued by OMT functions.

A certificate of type $U1$ is issued by functions $F_{bt}()$ and $F_{cat1}()$. The inputs to $F_{bt}()$ include a leaf node x in a subtree, the index i of the leaf node (in the sub-tree), and complementary nodes $\mathbf{v}$. The root of the subtree can now be computed as $y = f_m(x, i, \mathbf{v})$. The function also accepts another value x' and computes $y' = f_m(x, i, \mathbf{v})$ (using the same complementary nodes). The certificate of type $U1$ issued by this function, viz,

$$\rho = h(U1, x, y, x', y', \chi) \tag{9}$$

states that "(it has been verified by me that) y is the root of a sub-tree with leaf node x, and if $x \rightarrow x'$ then $y \rightarrow y'$." More generally, such a certificate implies that y is an *ancestor* of x, and that if $x \rightarrow x'$, then $y \rightarrow y'$.

Functions $F_{cat1}()$ and $F_{cat2}()$ combine self memoranda to issue (in general) more complex self-memoranda. $F_{cat1}()$ accepts inputs necessary to verify the integrity of two type $U1$ certificates. If the second certificate is 0, and if in the first certificate binding x_1, y, x'_1, y if $x_1 = x'_1$ (implying merely that y is an ancestor of x_1, a certificate of type $V1$, viz., $\rho = h(V1, x_1, y, \chi)$ is issued.

If the child in the second certificate x_2 is the same as the parent y in the first certificate, the two certificates are combined to issue a single certificate of type $U1$ binding the child x_1 in the first certificate to the parent z in the second certificate. Else, $F_{cat1}()$ computes $p = H_V(y, z)$ and $p' = H_V(y', z')$ to issue a certificate of type $U2$

$$\rho = h(U2, x_1, x_2, p, x'_1, x'_2, p', \chi) \tag{10}$$

to the effect that that "x_1 and x_2 are leaf nodes of a sub-tree with root p, and if $x_1 \rightarrow x'_1$ and $x_2 \rightarrow x'_2$ then $p \rightarrow p'$. Note that if y is an ancestor of x_1 and z is an ancestor of x_2, then $p = H_V(y, z)$ is simultaneously an ancestor of x_1 and x_2.

Function $F_{cat2}()$ extends the common ancestor y of two nodes to an ancestor z of y. In other words, $F_{cat2}()$ combines a $U2$ certificate with a $U1$ certificate to produce a $U2$ certificate. If only a certificate of type $U2$ is provided

$F_{sw}(x_1, x_2, y, y', \rho, o)$ {
IF $(\rho = h(U2, x_1, x_2, y, x_2, x_1, y', \chi))$
IF $(o = 1)$ RETURN $h(ER, y, y', \chi)$;
ELSE RETURN $h(EI, y, y', \chi)$;
}
$F_{ph}(A, A', \omega_A, B', y, y', \rho, o)$ {
IF $(\rho = 0) \wedge (o = 1)$ RETURN $h(ER, 0, H_L(A, A, 0), \chi)$
ELSE IF $(\rho = 0)$ RETURN $h(EI, 0, H_L(A, A, 0), \chi)$
$x_2 \leftarrow (o = 1)?\ H_L(A', B', \omega_A)\ :\ H_L(A', B', 0)$;
$x_1 \leftarrow H_L(A, A', \omega_A); x'_1 \leftarrow H_L(A, B', \omega_A); x'_2 \leftarrow 0$;
IF $((\rho = h(U2, x_1, x_2, y, x'_1, x'_2, y', \chi)) \vee$
$(\rho = h(U2, x_2, x_1, y, x'_2, x'_1, y', \chi)))$
RETURN $(o = 1)?\ h(ER, y, y', \chi)\ :\ h(EI, y, y', \chi)$;
}
$F_{ce}(i, x, y, \rho_1, z, \rho_2)$ {
IF $i \notin \{EI, ER\}$ RETURN;
IF $((\rho_2 = 0) \wedge (\rho_1 = h(i, x, y, \chi)))$ RETURN $h(i, y, x, \chi)$;
IF $((\rho_1 = h(i, x, y, \chi)) \wedge (\rho_2 = h(i, y, z, \chi)))$
RETURN $h(i, x, z, \chi)$;
}

Figure 3. OMT Functions for Issuing Equivalent-Root Memoranda.

as input to $F_{cat2}()$ with $x_1 = x'_1$ and $x_2 = x'_2$, bound to $y = y'$, $F_{cat2}()$ issues a certificate of type $V2$ binding two nodes x_1 and x_2 to a common ancestor y.

Certificates of type $U1$ and $U2$ are useful for simultaneously verifying and updating the root of the tree. Certificates of type $V1$ and $V2$ are useful in scenarios where only verification is required.

Functions $F_{ph}(), F_{sw}()$ and $F_{ce}()$ create certificates that bind equivalent roots. A certificate of $\rho = h(EI, y, y', \chi)$ attests to the equivalence of IOMT roots y and y'. A certificate $\rho = h(ER, y, y', \chi)$ attests to the equivalence of ROMT roots y and y'.

Through a certificate of type $U2$, $F_{sw}()$ recognizes the relationship between two roots resulting from swapping two leaves. As swapping leaves does not affect the integrity of an IOMT or an ROMT, the roots are equivalent for both IOMT and ROMT. Thus, depending on the value o which identifies the type of request ($o = 1$ for ROMT certificate) $F_{sw}()$ outputs a EI or ER certificate.

Function $F_{ph}()$ issues equivalence certificates binding roots before and after deletion of a place holder. The input $o = 1$ is a request to issue a ER certificate (else, the request is for an EI certificate). If no certificate is provided as input to $F_{ph}()$ (or $\rho = 0$), one root is assumed to the root of an empty tree, and the equivalent root is after insertion of the first place-holder for an index A. For both IOMT and ROMT the first place holder will be $(A, A, 0)$, and the root after insertion will be $H_L(A, A, 0)$.

If $\rho \neq 0$ this function interprets (A, A', ω_A) (with leaf hash x_1) and a place-holder (A', B', ω) (with leaf hash x_2) as two leaves in a tree with root y. If $o = 1$ (ROMT) the place holder has $\omega = \omega_A$, else (for an IOMT), $\omega = 0$. If the place holder is the first leaf it needs to be modified to (A, B', ω_A) (leaf-hash x'_1) and the second leaf to an empty leaf (leaf hash 0). The certificate ρ attests that modifying two leaves x_1 and x_2 to x'_1 and x'_2 is equivalent to changing the root from y to y'. Hence, y and y' are equivalent roots.

3. BGP SUBSYSTEMS

The Internet is an interconnection of autonomous systems (AS) [9], [10]. Each AS owns one or more chunks of the IP address space, where the number of addresses in each chunk is a power of 2. IP chunks are represented using the CIDR (classless inter-domain routing) IP prefix notation. For example, the IP prefix 132.5.6.0/25 represents 2^{32-25} IP addresses for which the first 25 bits are the same as the address 132.5.6.0, viz., addresses 132.5.6.0 to 132.5.6.127. An AS registry assigns AS numbers to AS owners. AS owners may acquire ownership of IP prefixes from an IP registry (through IP registrars, or ISPs).

While each AS may follow any protocol for routing IP packets within their AS, all ASes need to follow a uniform protocol for inter-AS routing. The current inter-AS protocol is the border gateway protocol (BGP), where AS owners employ one or more BGP speakers to advertise reachability information for IP prefixes owned by the AS. Specifically, every BGP speaker recognizes a set of neighboring BGP speakers. Neighbors may belong to the same AS or a different AS. The main responsibility of BGP speakers are

1. originate BGP update messages for prefixes owned by the AS, and convey such originated messages to neighbors

of other ASes

2. relay BGP update messages received from neighbors to other neighbors.

3. aggregate destination prefixes (that can be aggregated) for reducing the size of routing tables.

BGP is a path vector protocol. BGP update messages communicated between BGP speakers indicate an AS path vector for a prefix. Specifically, a BGP update message

$$[P_a, (A, B, C, D), W_d] \tag{11}$$

from a speaker S_d (belonging to AS D — the last AS in the path) indicates that prefix P_a owned by the first AS A in the path. W_d is the *weight* of the path.

3.1. BGP Updates

A BGP speaker may receive multiple paths for the same prefix. All such paths are stored by the BGP speaker in the incoming routing information database (RIDB-IN). However only the *best* path for a prefix may be copied to the outgoing database (RIDB-OUT), and advertised to other BGP speakers. Most often a BGP speaker is a component of a router which uses entries in RIDB-OUT (best path for different prefixes) to forward IP packets.

3.1.1. BGP Weights

The best path is the one with the maximum weight. Several parameters are used to compute the weight of a BGP path. For simplicity, in this paper we restrict ourselves to some of the more important weight parameters, i) pre-path weight; ii) local preference iii) AS path length; and iv) multi-exit descriptor (MED).

The pre-path weight is assigned at time of origination. If two paths for the same prefix have the same pre-path weight, then the the local preference is considered (higher the better). If both pre-path weight and local preference are the same, the AS path length (number of ASes in the path) is considered. The longer the path, the lower the weight. If the path lengths are also the same, then the MED weight is considered (higher the better).

Local Preference and MED: Every BGP speaker recognizes a set of other BGP speakers as neighbors. Every neighbor is associated with two weight parameters — a local preference, and an MED. From the perspective of a speaker S_a

1. L_b is the local preference of S_b implies that for all paths received *from* S_b the local preference component of the weight should be reset to L_b.

2. M_b is the MED of S_b implies that for all paths advertised *to* S_b, the MED component of the weight should be set to M_b.

Local preference and MED weights are assigned only to neighbors that are speakers of foreign ASes.

Processing Received BGP Updates: When a BGP update message is received from a foreign speaker S_b (of AS B) the steps to be taken by a speaker S_a (AS A) are as follows:

1. Increment hop-count;

2. Add own AS A to the path vector;

3. Change local preference to value L_b;

4. Set next hop to S_b

5. Store path in RIDB-IN.

When a path is received from a speaker S'_a belonging to the *same* AS, no component of the weight is changed, and the AS number is not inserted (as it was already inserted by the previous hop). The next hop is set to S'_a.

Relaying and Originating BGP Updates: For relaying a BGP message for a prefix P to a BGP speaker S_b in a foreign AS, the steps to be taken by speaker S_a are

1. Among all paths for the same prefix, choose the path with the highest weight

2. Change the MED component of weight to M_b;

3. Advertise the path with modified weight.

For originating a path (for owned prefixes), the pre-path weight is set, and the MED is set to that of the foreign neighbor. Such originated paths are *not* sent to speakers of the same AS (as paths to IP addresses within the AS are established using an intra-AS protocol). For relaying a BGP update message (for a prefix owned by a foreign AS) to a speaker S'_a of the same AS, simply choose the path with the highest weight and send it without changing the weight.

Policies and Preferences: The choice of BGP speakers for the AS, the prefixes for which a speaker may originate BGP update messages (along with their pre-path weights), neighbors of each speaker, along with their local preference and MED weights, etc., can be seen as policies and preferences specified by the AS owner to influence the weights assigned to BGP paths.

Aggregation: One of the major benefits of CIDR prefixes come from the fact that BGP speakers may aggregate prefixes. If two consecutive prefixes A and B (say 126.5.4.0/25 and 126.5.4.128/25) and can be aggregated into a single prefix C (126.5.4.0/24) if the next hop for prefixes A and B is the same. The speaker that performed the aggregation is the originator for the aggregated prefix.

4. SECURITY KERNELS FOR BGP SUB-SYSTEMS

Thus far we have outlined generic security kernel functionality for issuing OMT certificates. In this section we consider other sub-system specific security kernel functionality for various BGP sub-systems like AS and IP registries, AS owners, and BGP speakers.

For simplicity, we shall assume a single registry for both AS numbers and IP addresses. All security kernel modules have a unique identity. Let U be the identity of the module associated with the registry. One module is associated with every AS owner. We shall assume the identities of an AS owner modules to be the same as the AS number. Each BGP speaker is associated with a module. We shall assume that the identity of BGP speaker modules to be the IP address of the router/BGP speaker. We also assume the existence of module functionality for authentication/verification of messages exchanged between modules. Specifically, we shall represent such functionality as

$$\begin{aligned} \mu &= f_a(X, Y, \{v_1, v_2, \ldots\}) \text{ and} \\ \{0,1\} &= f_v(X, Y, \{v_1, v_2, \ldots\}, \mu) \end{aligned} \tag{12}$$

the process of authentication (by module X, using $f_a()$) and verification (by module Y, using $f_v()$) of a message conveying values $\{v_1, v_2, \ldots\}$, from module X to module Y. Function $f_a()$ outputs a authentication code μ. Function $f_v()$ outputs a binary value (TRUE if authentication μ is consistent, or FALSE).

The identity U of the registry module is known to all AS owner modules. The registry module U delegates AS numbers and IP prefixes to AS owner modules. AS owner modules will only accept delegations from U. AS owner modules in turn delegate IP address ranges they own to one or more BGP speaker modules.

Some of the specific desired assurances regarding the operation of BGP are as follows:

1. an AS number can not have more than one owner; an IP address can not be owned by one or more ASes. Such assurances should be guaranteed even if the computers employed by the registry have been compromised by an attacker.

2. AS owners can only delegate address ranges owned by the AS to BGP speakers.

3. Notwithstanding the possibility that a router/ BGP speaker may be under the control of an attacker, the following assurances are desired

   (a) The BGP speaker will only be able to create BGP update messages for prefixes delegated by the AS owner

   (b) No BGP update message can be created by violating any of the policies / preferences specified by the AS owner (neighboring speakers, local preference and MED, pre-path weights) or BGP rules (only the path with the best weight can be advertised).

   (c) A speaker will not accept paths which already includes its own AS (to ensure that routing loops can not be created).

   (d) All BGP speakers will increment the hop count exactly by one.

   (e) A speaker will be able to aggregate only prefixes for which the next hop is the same speaker.

$F^R_{ph}(x, \rho)$ {
IF $(\rho = h(ER, \xi_r, x, \chi))$ $\xi_r \leftarrow x$;
}
$F^R_{as}(I, I', A, \rho, \xi'_r)$ {
IF $(\rho = h(U1, H_L(I, I', 0), \xi_r, H_L(I, I', A), \xi'_r, \chi)$ $\xi_r \leftarrow \xi'_r$;
}
$F^R_{dp}(I, I', A, \rho)$ {
IF $(\rho = h(V1, H_L(I, I', A), \xi_r, \chi))$ RETURN $f_a(U, A, \{x\})$;
}
$F^O_{ph}(x, \rho)$ {
IF $(\rho = h(ER, \xi_r, x, \chi))$ $\xi_p \leftarrow x$;
}
$F^O_{ap}(I, I', \mu, \rho, \xi'_r)$ {
$x \leftarrow H_L(I, I', 0); x' \leftarrow H_L(I, I', A)$;
IF $(f_v(U, A, x', \mu) = 0)$ RETURN;
IF $(\rho = h(U1, x, \xi_r, x', \xi'_p, \chi)$ $\xi_r \rightarrow \xi'_r$;
}
$F^O_{dp}(\xi'_o, \rho, S, \xi'_n)$ {
IF $(\rho = h(V1, \xi'_o, \xi_r, \chi)$ RETURN $\mu = f_a(A, S, \{\xi'_o, \xi'_n\})$;
}

Figure 4. Security Kernel Functionality in Registry and AS Owner Modules.

4.1. OMTs Used by BGP Subsystems

The registry and AS owners maintain an ROMT where each leaf indicates a range of IP addresses, and the third value is the AS number (of the AS that owns the address range).

BGP speakers maintain one ROMT, multiple IOMTs, and a plain merkle tree. A plain merkle tree is used to maintain a neighbor table with a static[2] number of records. The ROMT is used maintaining address ranges for which the speaker can originate BGP updates (owned prefixes and aggregated prefixes).

An IOMT is used for maintaining the RIDB-IN database. More specifically a nested IOMT is used where the root corresponds to a tree with leaves whose indexes are IP prefixes. Corresponding to each prefix the value (third field) is the hash of two IOMT roots θ and γ. The root θ of the "path tree" has one leaf for every path for the prefix. The root γ of the "weight tree" represents the weights of different paths, and enables the module to readily identify the path with the highest weight. The index of leaves in path tree is a function of a quantity α that is itself the root of an IOMT. Specifically, the "path vector" IOMT with root α has a leaf corresponding to every AS in the AS path. Representing the AS path in this way makes it possible for the module to recognize that it is already in the path, and thereby prohibit creation of routing loops.

4.2. Registry Module U and AS Owner Modules

The registry module maintains an ROMT root ξ_r, where each leaf indicates ranges of IP addresses, and the AS number of the owner. Unassigned IP chunks have a leaf with (third) value 0.

The function $F^R_{ph}()$ can be utilized to insert/delete any place holders in the ROMT by providing a memoranda of type ER.

The registry employs the function $F^R_{as}()$ to convert the third value of any leaf from 0 to a non zero value.

A leaf (I, I', A) in the ROMT indicates that the IP addresses in the range I and $I' - 1$ have been assigned to AS A. The leaf (I, I', A) can be conveyed to an AS owner module A using interface $F^R_{dp}()$.

AS owner modules also maintain an ROMT with root ξ_r. The leaves indicate IP addresses owned by the AS. In the tree maintained by the owner of AS A who (for example) owns two non consecutive chunks with addresses between $[a, a')$ and $[b, b')$ the ROMT leaves will be (a, a', A), $(a', b, 0)$, (b, b', A) and $(b', a, 0)$.

The function $F^o_{ph}()$ can be used to insert/delete place-holders in the tree. Once a place older $(a, a', 0)$ exists, a delegation (a, a', A) from the registry module U can be used to update the place holder to a leaf (a, a', A).

Any node in the tree with root ξ_r can now be sub-delegated to a BGP speaker. Depending on which prefixes need to be delegated to which BGP speaker the owner can use $F^O_{ph}()$ to subdivide owned prefixes and swap positions of prefix leaves, and choose the root of a subtree which includes all prefixes to be delegated to the speaker.

[2]For scenarios involving dynamic databases where records can not be inserted or deleted (the dynamics come only from modification of records) OMT is an over-kill; a plain merkle tree is adequate.

Apart from delegating IP prefixes, the AS owner also specifies various preferences as leaves of a hash tree (with root ξ'_n). The types of records in this tree include

1) Pre-path weight; a record of the form $[P, o]$ for each owned prefix P that can be originated by the speaker, indicating the pre-path weight o.

2) Neighbor preferences record for each neighbor. A record for neighbor F is of the form

$$\mathbf{N}_F = [F, s_f = 0, t_f = 0, A_f, L_f, M_f, \tau_f] \tag{13}$$

where A_f is the AS number of the neighbor, L_f and M_f are the local preference and MED weights, and τ_f is the maximum permitted duration between HELLO messages from the neighbor N.

The values s_f and t_f are set to zero by the AS owner. Such fields can be modified only by the module of a BGP speaker initialized using the value ξ'_n. The value s_f is the time at which a link to F was established. Value t_f is the time at which the F was last heard-from.

4.3. BGP Speakers

The security kernel of BGP speakers maintains 3 dynamic roots (see Figure 5)

1. the root ξ_o of an ROMT. This is initialized to a value ξ'_o communicated by the AS owner module.
2. the root ξ_n of a merkle tree with a leaf corresponding to every neighbor, and a static leaf for every owned prefix corresponding to which the BGP speaker can originate BGP updates. This root is initialized to the value ξ'_n conveyed by the AS owner module.
3. the root ξ_d, an IOMT indexed by IP prefix — the RIDB tree. This value is initialized to zero.

BGP speakers also maintain a static value A — initialized to the AS number represented by the speaker.

During regular operation of the BGP speaker the RIDB root ξ_d is updated whenever a BGP update message is received, or if a path is removed (for example) due to loss of link to neighbor.

The neighbor/preferences tree root ξ_n is updated whenever a neighbor state is updated. Specifically, corresponding to each neighbor are two dynamic values: a connection identifier s (which is the time at which the connection was initiated) and a time-stamp t (time of last activity in the connection).

The leaves of the ROMT are IP address ranges for which the speaker can *originate* BGP updates. Originated updates can be for owned IP address ranges or for aggregated prefixes. When initialized, the ROMT root ξ_o is a commitment to leaves corresponding to owned IP ranges (delegated by the AS owner module by conveying a root of a sub-tree from its tree of owned prefixes). In all such leaves the third value a is the AS number. The ROMT root ξ_o may also be updated for purposes of aggregating CIDR prefixes. Specifically, for any prefix in the RIDB tree the address range and the next hop in the best path to the prefix can be added to the ROMT. Thus, for leaves corresponding to foreign IP ranges the third value is the next hop. Two adjacent prefixes with the same next hop can now be aggregated. More specifically, aggregation corresponds to removing a place-holder. For example, two leaves (I_1, I_2, x) and (I_2, I_3, x) where $[I_1, I_2)$ and $[I_2, I_3)$ are two ranges with the same next hop x, can be converted to a single leaf (I_1, I_3, x) through an equivalence operation.

From the perspective of the BGP speaker modules, corresponding to a BGP update message from a speaker (with IP address) X to a speaker Y is an authenticated message from module X to module Y computed as

$$\mu = f_a(X, Y, \{P, \alpha, l, w_{pp}, w_{lp}, w_{med}]\}) \tag{14}$$

where P is the prefix for which the path is advertised, α is a one-way function of the AS path, l is the path length, w_{pp}, w_{lp} and w_{med} are respectively the pre-path weight, local preference, MED. The four weights are used to construct a weight represented as

$$W = [w_{pp} \parallel w_{lp} \parallel MAX - l \parallel w_{med}]. \tag{15}$$

Thus, for any prefix the path with the highest weight W is the best path.

Security kernel functions $F^S_{rel}()$ and $F^S_{orig}()$ are used to create such BGP update messages, and $F^S_{upd}()$ is used to process such messages from neighboring speakers and update the RIDB root. More specifically

1) $F^S_{orig}()$ is used to originate BGP updates (for own prefixes and aggregated prefixes). Specifically, a path for a prefix P (represented in the origin tree as a leaf with range $[I_1, I_2)$ and third value v) can be advertised only if

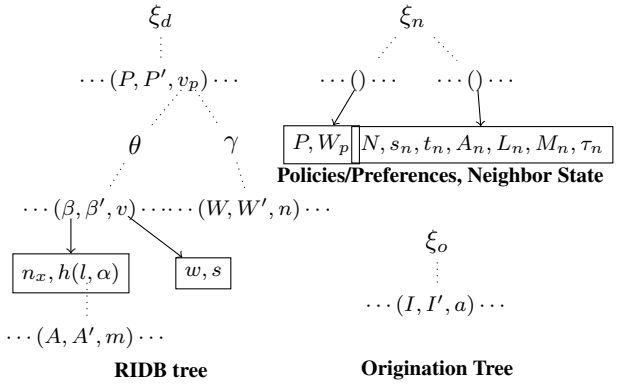


Figure 5. OMTs Used by BGP Speakers.

1. the third value v is its own AS number, *and* a leaf exists in the tree with root ξ_n for the prefix P, conveying the pre-path weight w_{pp} for prefix P; or

2. the third value v corresponds to a neighbor with a live link, *and* no leaf with prefix P exists in the RIDB tree.

2) $F^S_{rel}()$ is used to relay stored BGP paths in the RIDB to neighbors. $F^S_{rel}()$ identifies the best path for a prefix, and only the best path may be advertised. Alternately, information regarding the best path can also be added to the origination tree to aggregate a prefix.

Neighboring BGP speakers maintain a TCP connection over which BGP update messages are exchanged. To keep the connection alive, and for testing the existence of the link, special HELLO messages are exchanged periodically. From the perspective of the security kernel in a speaker S the link to a neighbor F is associated with the link establishment time s_f and a time-stamp t_f. Once a link has been established, the module in F is expected to confirm their continued presence by periodically sending authenticated time-stamped messages for updating the time-stamp t_f.

In the RIDB-IN, multiple paths, each with possibly different weights, may exist for each prefix. To enable the security kernel to readily determine the path with the highest weight, the plurality of weights for each prefix are maintained as an ordered list.

In the weight IOMT, the index of a leaf is a weight, and the value (third field) is the number of occurrences of the weight in the list. For example, corresponding to a list with four weights $(21, 21, 34, 42)$, three leaves $(21, 34, 2), (34, 42, 1), (42, 21, 1)$ will exist in the weight tree (index 21 occurs twice as indicated by the value field). As in any IOMT, insertion of a place holder (say for index 5, which signifies "zero occurrences of value 5 in the list") does not modify the list.

Within the RIDB IOMT a special IOMT is also used to represent AS paths. In the AS path IOMT the the index of leaves are ASes. A tree corresponding to a path of length 5 will have 5 leaves. The value field (third field) is the position in the path. As an example, corresponding to a path $A \rightarrow D \rightarrow B \rightarrow E$ the leaves of the tree will be $(A, B, 1), (B, D, 3), (D, E, 2)$ and $(E, A, 4)$ (note that the value for index D is 2 as D is the second AS in the path).

In the RIDB IOMT the index of leaves are IP prefixes. The value field in the IOMT is a one way function of two IOMT roots

1. IOMT root γ — is the root of a weight-IOMT; and

2. IOMT root θ — the root of an IOMT whose leaves like (β, β', v) characterize each path to the prefix.

In the IOMT with root θ

1. the index of leaves are functions of the path; more specifically, in the index $\beta = h(G, h(l, \alpha))$, G is the next hop, l is the path length, and α is the root of an AS-path IOMT root.

2. The value v corresponding to an index β is a function of two values — the weight W of the path, and the connection identifier of the next hop that provided the path. If the connection identifier in a path is not the same as the identifier in the neighbor record for that neighbor, then the path is considered as stale (and the weight is set to 0).

$F^S_{init}(A', \xi'_o, \xi'_n, \mu)$ {
IF $(f_v(A', S, \{\xi'_o, \xi'_n\}, \mu))$
$\xi_n \leftarrow \xi'_n$; $\xi_o \leftarrow \xi'_o$; $\xi_d \leftarrow 0$; $A \leftarrow A'$;
}
$F^S_{ph}(\xi', \rho, o)$ {
IF $(o = 1) \wedge (\rho = h(ER, \xi_o, \xi', \chi))$ $\xi_o \leftarrow \xi'$;
ELSE IF $(o = 2) \wedge (\rho = h(EI, \xi_r, \xi', \chi))$ $\xi_d \leftarrow \xi'$;
ELSE IF $(o = 3) \wedge (\rho = h(E2, \xi_r, \xi', \chi))$ $\xi_d \leftarrow \xi'$;
}
$F^S_{ph2}(\theta, \theta', \rho_1, \gamma, \gamma', \rho_2, P, P', \rho, \xi, \xi')$ {
IF $(\rho_1 \neq 0) \wedge (\rho_1 \neq h(EI, \theta, \theta', \chi))$ RETURN;
IF $(\rho_1 \neq 0) \wedge (\rho_1 \neq h(EI, \gamma, \gamma', \chi))$ RETURN;
IF $(\rho_1 = 0)$ $\theta' \leftarrow \theta$;
IF $(\rho_2 = 0)$ $\gamma' \leftarrow \gamma$;
$v_p \leftarrow H_L(P, P', h(\theta, \gamma))$; $v'_p \leftarrow H_L(P, P', h(\theta', \gamma'))$;
IF $(\rho = h(U1, v_p, \xi, v'_p, \xi', \chi))$ RETURN $h(E2, \xi, \xi', \chi)$;
}
$F^S_{hlo}(G, s_g, t_g, A_g, L_g, M_g, \tau_g, \rho, \xi'_n, s'_g, t'_g, \mu, t')$ {
$v \leftarrow h(G, s_g, t_g, A_g, L_g, M_g, \tau_g)$;
IF $(\mu = 0) \wedge (|t - t'| < \delta)$ $\{s'_g = t', t'_g = 0\}$;
ELSE IF $(\mu = 0) \wedge (t' = 0)$ $\{s'_g = 0, t'_g = 0;\}$
ELSE IF $(\mu = 0)$ $\{s'_g = s_g, t'_g = t_g;\}$
ELSE IF $(f_v(G, S, \{s'_g, t'_g\}, \mu) = 0)$ RETURN :
IF $((s'_g < s_g) \vee ((s'_g = s_g) \wedge (t'_g < t_g)))$ RETURN :
IF $(\rho = h(U1, v, \xi_n, h(G, s'_g, t'_g, A_g, L_g, M_g, \tau_g), \xi'_n, \chi))$
$\xi_n \rightarrow \xi'_n$; RETURN$f_a(S, G, \{s'_g, t\})$;
}

Figure 6. Security Kernel Functions for BGP Speakers.

4.4. Using Security Kernel Functions in BGP Speaker Module

BGP speaker modules expose a function $F^S_{init}()$ which is invoked to initialize the module. In the rest of this paper we shall investigate the functionality of a speaker S belonging to an AS A. An authenticated message from AS module A (created by using function $F^A_{dp}()$ in Figure 6) is necessary for initializing the roots of the neighbor tree to ξ'_n, and the origin tree to ξ'_o.

Any place holder can be added to the IOMT with root ξ_r or the ROMT with root ξ_p. Using function $F^S_{ph}()$. Any place holder can also be added to the path tree or weight tree corresponding to any prefix. This can be accomplished using function $F^S_{ph2}()$ which issues a equivalence memoranda of type $E2$ identifying two roots corresponding to before and after insertion of a place holder in a tree with root θ, or a tree with root γ, or both.

Function $F^{hlo}_S()$ can be invoked to create authenticated messages that can be sent to other speakers. This function ensures that speaker S can only connect to speakers explicitly authorized by the AS owner (by providing the initial root ξ_n). Such authenticated messages can be used to create a connection (with a new value of s deemed sufficiently close to the current time t), and for updating time stamps of neighbors.

4.5. Processing BGP Updates

Function $F^S_{upd}()$ is invoked to update the RIDB-IN tree — either due to a BGP update message received from a neighbor, or due to loss of link to the next hop. From the perspective of the security kernel the link to the next hop is broken if the time-stamp in the neighbor record is stale. If the current neighbor session identity is different from the session identity of the next hop in the stored path, then the path is assumed to be invalid (as the path was provided during an earlier session). If the neighbor is no longer active, or if the path is invalid, the path weight will be set to 0.

$F^S_{upd}()$ is invoked to update a path for a prefix P. Recall that a prefix P is associated with a path tree root θ and a weight tree root γ. A path in the path tree is uniquely identified as a function of the AS-path α, path-length l, and next hop N: the index of the path is $\beta = h(N, h(l, \alpha))$. The path is associated with a path weight W_c and the session identity s_n of the next hop.

Updating the path implies modifying the current weight W_c associated with the index β to a weight W. In addition, modification of the weight requires the weight tree to be modified. Specifically

1. if $W_c = 0$ and $W \neq 0$ (inserting a path), then the value W has to be added to the IOMT with root γ;

$F^S_{upd}([P, \alpha, l, w_{pp} \parallel w_{lp} \parallel w_{med}], \mu,$ // Update regarding prefix P
$\rho_n, \mathbf{N}_N = [N, s_n, t_n, A_n, L_n, M_n, \tau_n],$ // from neighboring speaker N
$\beta', (W_c, s_c), \theta, \theta', \rho_t,$ // insert path in path tree
$\gamma, W', m, \gamma', \rho_w,$ // and weight in weight tree
$P', \rho_d, \xi'_d)$ //and update RIDB root $\xi_r\{$
IF $(\rho_n \neq h(V1, h(\mathbf{N}_N), \xi_n, \chi))$ RETURN;
IF $(\mu = 0) \wedge ((s_n \neq s_c) \vee (t > l_n + \tau_n))$ $W \leftarrow 0$;
ELSE IF $(\mu = 0) \wedge (W_c = 0)$ $W \leftarrow -1$;
ELSE IF $(f_v(N, S, \{s_n, P, \alpha_n, w_{pp} \parallel w_{lp} \parallel w_{med}\}, \mu) = 1)$
IF $(A \neq A_n)$ $W \leftarrow [w_{pp} \parallel L_n \parallel MAX - l \parallel w_{med}]$;
ELSE $W \leftarrow [w_{pp} \parallel w_{lp} \parallel MAX - l \parallel w_{med}]$;
$v \leftarrow (s_c = 0)?0 : h(W_c, s_c); v' \leftarrow (W = -1)?0 : h(W, s_n);$
$\beta = h(N, h(l, \alpha)); v \leftarrow H_L(\beta, \beta', v); v' \leftarrow H_L(\beta, \beta', v');$
IF $(\rho_t \neq h(U1, v, \theta, v', \theta', \chi))$ RETURN;
IF $((W \leq 0) \wedge (m > 0) \wedge (W_c > 0))$
$v \leftarrow H_L(W_c, W', m); v' \leftarrow H_L(W_c, W', m - 1);$
ELSE IF $((W_c = 0) \wedge (W < 0))$ $v \leftarrow v' \leftarrow 0$;
ELSE IF $(W > 0)$ $v \leftarrow H_L(W, W', m); v' \leftarrow H_L(W, W', m + 1);$
IF $(\rho_w \neq h(U1, v, \gamma, v', \gamma', \chi)))$ RETURN;
$v \rightarrow H_L(P, P', h(\alpha, \gamma)); v' \rightarrow H_L(P, P', h(\alpha', \gamma'));$
IF $\rho_d = h(U1, v, \xi_d, v', \xi'_d, \chi)$ $\xi_d \leftarrow \xi'_d$;
}

Figure 7. BGP Speaker Security Kernel Functionality for Accepting BGP Updates.

2. if $W = 0$ and $W_c \neq 0$ (setting path weight to 0) then the value W_c has to be removed from the tree with root γ.
3. if $W = W_c = 0$, then $F^S_{upd}()$ is invoked to delete a path with zero weight. In this case no change is necessary to the weight tree root γ.

For inserting a path $F^S_{upd}()$ is invoked by submitting a received BGP update from a neighbor N specifying path vector α, path length l, and weights $w_{pp} \parallel w_{lp} \parallel w_{med}$. The weight for the inserted path is then

$$W = w_{pp} \parallel x \parallel MAX - l \parallel w_{med} \tag{16}$$

where $x = L_n$ or $x = w_{lp}$. Specifically, if the neighbor N providing the update belongs to from a foreign AS, the $x = L_n$ (the local preference of X); if N belongs to the same AS, the local preference w_{lp} advertised by N is retained.

For setting weight to zero $F^S_{upd}()$ may be invoked without a BGP message, or a BGP message that withdraws a previously advertised path. A withdraw message from a neighbor indicates $w_{pp} = w_{lp} = w_{med} = 0$.

In general, updating a path with index β (for prefix P) will require modification to the path tree root θ and weight tree root γ (in the leaf for prefix P). For incorporating the change in values α and γ associated with leaf index P, the RIDB root ξ_d will need to be modified.

The inputs to $F^S_{upd}()$ include

1. a neighbor record $\mathbf{N}_N$ for N and a $V1$ memoranda ρ_n to verify the integrity of the record against the root ξ_n.
2. $U1$ memoranda ρ_t, necessary to update a leaf with index β in a tree with root θ,
3. $U1$ memoranda ρ_w, necessary to increment the counter in leaf with index W (when a path with weight W is inserted), or decrement the counter in a leaf with index W_c (when the current weight W_c of the path is reset to 0), in the weight tree with root γ,
4. $U1$ memoranda ρ_d, necessary to update the RIDB-IN root ξ_d due to the changes to values γ and θ associated with index P; and
5. a received authenticated BGP update message $[\alpha, l, w_{pp} \parallel w_{lp} \parallel w_{med}, \mu]$ from neighbor N.

4.6. Advertising BGP Paths

Function $F^S_{adv}()$ is invoked to identify the best path for a prefix and a) advertise the best path (create BGP update) to a neighbor, or b) add the prefix for the path (along with the next hop and session identity of the next hop) to the origination[3] tree. $F^S_{adv}()$ can also be invoked to create a BGP update to withdraw a path with weight 0;

[3]This is to enable aggregation of prefixes. Two adjacent prefixes with the same next hop and session identity can be aggregated by removing a place-holder in the ROMT.

$F^S_{snd}(\alpha, l, \beta', W, \theta, \rho_t, W', m, \gamma, \rho_w, P, P', \rho_d,$
$\rho_n, \mathbf{N}_G, \alpha_i, \rho_i, \alpha', \rho_{as}, \rho_f, \mathbf{N}_F, \rho_o, \xi'_o)\{$
IF $((F = 0) \wedge (\rho_n \neq h(V1, h(\mathbf{N}_G), \xi_n, \chi)))$ RETURN;
ELSE IF $(\rho_n \neq h(V2, h(\mathbf{N}_F), h(\mathbf{N}_G), \xi_n, \chi))$ RETURN;
$\beta \leftarrow h(G, h(l, \alpha)); v \leftarrow H_L(\beta, \beta', h(W, s_g))$
IF $(\rho_t \neq h(V1, v, \theta, \chi))$ RETURN;
IF $(\rho_w \neq h(V1, H_L(W, W', m), \gamma, \chi))$ RETURN;
IF $(m < 1) \vee (W > W')$ RETURN ;
IF $(\rho_d \neq h(V1, H_L(P, P', h(\theta, \gamma)), \xi_d, \chi))$ RETURN;
IF $(F = 0)$//let $[x, y)$ is the address range of prefix P
$P \rightarrow [x, y); v \leftarrow (x, y, 0); v' \leftarrow (x, y, G \parallel s_g);$
IF $(\rho_o = (U1, v, \xi_o, v', \xi'_o, \chi))$ $\xi_o \leftarrow \xi'_o$; RETURN;
IF $(A_f \neq A)$
IF $(\rho_i \neq h(EI, \alpha, \alpha_i, \chi))$ RETURN;
$l \leftarrow l + 1; v \leftarrow H_L(A, A', 0); v' \leftarrow H_L(A, A', l);$
IF $(\rho_{as} \neq h(U1, v, \alpha_i, v', \alpha'))$ RETURN ;
ELSE $(\alpha' \leftarrow \alpha)$;
IF $(W = 0)$ RETURN $f_a(S, F, \{s_f, \alpha', l, 0 \parallel 0 \parallel 0\})$;
ELSE IF $((t < t_f + \tau_f) \wedge (t < t_g + \tau_g))$
$[w_{pp}, w_{lp}, w_{len}, w_{med}] \leftarrow W;$
IF $(A = A_f)$ RETURN $f_a(S, F, \{s_f, P, \alpha, l, w_{pp}, w_{lp}, w_{med}\})$
ELSE RETURN $f_a(S, F, \{s_f, P, \alpha', l, w_{pp} \parallel 0 \parallel M_f\})$
}

Figure 8. BGP Speaker Security Kernel Functionality for Relaying BGP Updates.

If W is the best weight for prefix P then a leaf (W, W', m) with $W' < W$ and $m \neq 0$ should exist in the tree with root γ. This is demonstrated using a memorandum of type $V1$. There should also exist a leaf for an index $\beta = h(G, h(l, \alpha))$ in the tree with root θ associated with values s_g and W. For this purpose a $V1$ memoranda is necessary to demonstrate the integrity of a neighbor record for G against ξ_n, and another $V1$ memoranda is required to demonstrate the integrity of the leaf with index β against root θ. Finally, another $V1$ memoranda is required to demonstrate the integrity of values θ and γ associated with index P against the RIDB root ξ_d.

Now that the best path (described by next hop G, AS vector α, path length l and weight W) has been identified,

1. a leaf with range $[x, y)$ corresponding to prefix P can be added to the origination tree indicating next hop and session identity $G \parallel s_g$, or
2. a BGP update for prefix P can be created and sent to a neighbor F.

In the former case, updating the origination tree will require a leaf $(x, y, 0)$ to be modified to $(x, y, G \parallel s_g)$ where $P \equiv [x, y)$. For updating the leaf of the origination tree, a $U1$ memoranda is required as input to $F^S_{adv}()$.

Before a BGP message for a path can be advertised to a *foreign* neighbor F, the path vector and path length have to be modified (to insert own AS number). If the path vector root is currently α, and the length is currently l, the value l should be incremented, and a new leaf needs to be inserted into the IOMT with root α. Specifically, the new leaf will have index A (AS number of the speaker) and value $l + 1$. More specifically, a place holder for A needs to be inserted in a tree with root α, following which the place holder can be updated to modify the third field from 0 to $l + 1$. Thus, a memoranda of type EI (for inserting a place holder) and a memoranda of type $U1$ (for updating the place-holder) are required as inputs.

4.7. Originating BGP Updates

$F^S_{orig}()$ is used to advertise path information for two categories of prefixes 1) prefixes owned by the AS; and 2) aggregated prefixes. Specifically, in leaves corresponding to owned prefixes in the origination tree, the third value will be its own AS number A. Corresponding to other leaves the third value will be a neighboring speaker G (next hop for the prefix) and a session identity s'_g of G (at the time the prefix was added to the origination tree).

An owned range $[x, y)$ can be converted into a prefix P and advertised to a neighbor F only if a record (P, w_{pp}) exists in the neighbor/policies tree with root ξ_n. A certificate of type $V2$ is provided as input to simultaneously verify the integrity of the neighbor record $\mathbf{N}_F$ and record (P, w_{pp}) in the neighbor tree.

To advertise an aggregated prefix P a $V1$ memoranda attesting the integrity of the next hop neighbor record $\mathbf{N}_G$ is required. In addition, a $V1$ certificate is required to demonstrate that prefix P does *not* exist in the RIDB-IN

$F^S_{orig}(P, P', W_p, \rho_o, \rho_r, \rho_n, \mathbf{N}_F, \rho_f, \mathbf{N}_G, s'_g, \xi'_o)\{$
IF $(A_f = A)$ RETURN;
IF $(G = 0)$ $v \leftarrow A$; ELSE $v \leftarrow G \parallel s'_g$;
$P \rightarrow [x, y); v \leftarrow H_L(x, y, v)$;
IF $((F = 0) \wedge (s_g \neq s'_g))$ // Remove aggregated prefix
  IF $(\rho_n \neq h(V1, h(\mathbf{N}_G), \xi_n, \chi)))$ RETURN;
  IF $(\rho_o = h(U1, v, \xi_o, H_L(x, y, 0), \xi'_o, \chi)$ $\xi_o \leftarrow \xi'_o$; RETURN;
IF $(t > t_f + \tau_f)$ RETURN ;
IF $(\rho_o \neq h(V1, v, \xi_o, \chi)$ RETURN;
IF $(G = 0) \wedge (\rho_n = h(V2, h(P, W_p), h(\mathbf{N}_F), \xi_n, \chi))$
  RETURN $f_a(S, F, \{s_f, P, H_L(A, A, 1), 1, W_p \parallel 0 \parallel M_f\})$
IF $(\rho_n \neq h(V2, h(\mathbf{N}_F), h(\mathbf{N}_G), \xi_n, \chi)))$ RETURN;
IF $(\rho_r \neq h(V1, H_L(P, P', 0), \xi_r, \chi))$ RETURN;
IF $(t < t_g + \tau_g)$
  RETURN $f_a(S, F, \{s_f, P, H_L(A, A, 1), 1, 0 \parallel 0 \parallel M_f\})$
}

Figure 9. BGP Speaker Security Kernel Functionality for Originating BGP Updates.

tree.

If the next hop F (to whom the origination message is to be sent) is set to $F = 0$, then $F^S_{orig}()$ interprets this as a request to delete an aggregated leaf for prefix P with third value $G \parallel s'_g$. To remove the aggregated prefix the third value $G \parallel s'_g$ is set to 0. For this purpose a certificate ρ_o of type $U1$ is required as input.

When a BGP message is originated for an owned prefix or an aggregated prefix the MED weight is set to to value M_f (for the intended receiver F) provided by the AS owner; the local preference is set to 0; for owned prefixes the pre-path weight is set to the value W_p prescribed by the AS owner, and for aggregated prefixes the pre-path-weight is set to 0.

5. RELATED WORK AND CONCLUSIONS

In a large majority of security-kernel based approaches in the literature, the purpose of the security kernel is to ensure that verified software is executed unmolested on an untrusted platform. In the trusted computing group (TCG) approach based on the trusted platform modules (TPM) only the security kernel is trusted to realize the assurance that that "only pre-verified software can take control of the platform."

The security kernel, or the TCB for the TCG-TPM approach, can be seen as composed of three *roots of trust* — the root of trust for storage (RTS), reporting (RTR) and measurement (RTM). The RTS and RTR are offered by a hardware TPM bound to an untrusted platform. The RTM includes "all essential hardware required to run software." Most often, the "essential hardware" includes the CPU, RAM, CPU-RAM bridge and BIOS.

The purpose of the RTM is to measure every unit of software that takes control of the CPU. The unit of software is typically a file, and the measure is the file hash. The trusted BIOS includes software that measures itself, reports the measurement to the TPM, load the next level of software (usually the boot-loader), measure the boot-loader, and report the measurement to the TPM.

If the boot loader can be verified to be free of malicious code then the boot loader loads the next level of code (the operating system kernel), measures the kernel and reports the measurement to the TPM. Similarly the operating system can load other higher level components and report measurements to the TPM.

The RTS is trusted to securely store measurements; the RTR is trusted to report measurements. Any entity interacting with the untrusted platform can now request the TPM to report the measurements, and may choose to abandon the interaction if the reported measurements differ from expected measurements. This strategy of building a chain of trust starting with the BIOS is the AEGIS model [11] adopted in the TCG approach.

The main issues with the TCG-TPM approach are three fold:

1. Ensuring that software can run unmolested is very little comfort when the software itself becomes too complex to be thoroughly verified. Furthermore, hidden malicious functionality in complex software may actually load other software without reporting their measurements (or reporting arbitrary measurements) to TPM.

2. Lack of a secure binding between the RTM (trusted components of the untrusted platform) and the TPM (which houses RTS and RTR). The implication of this is that the TPM can uncoupled from the RTM, and supplied expected measurements (while the platform runs arbitrary software).

3. The "minimal hardware trusted to run software" may also include peripherals with direct access to RAM. This results in the well known TOCTOU problem [12] in the TCG approach.

In the proposed approach the goal of the security kernel is not to ensure the integrity of the all software related to a system/ sub-system. Rather, the goal is to ensure only some very specific sub-system specific properties. For example, if the TCG approach is used to secure the AS/IP Registry, every computer used by the Registry should be TPM enabled, and every piece of software that can take control of any computer should be carefully examined to be free of malicious code. However, in the proposed approach, only the simple security kernel functionality outlined in Figure 4 needs to be assured to be clear of undesired functionality.

Most commonly used hash tree based ADSs include the well known merkle tree [8], skip-lists [7], red-black trees [5], and B-trees [3, 13]. All such ADSs (except the plain Merkle tree) essentially provide the capability to *order* values in a set (based on some index). The main difference between OMTs and other ADSes like skip-list, red-black trees and B-trees are

1. In the OMT, the ordering is virtual (the first two fields in an OMT can be seen as a circular link list). In other trees the ordering is physical.

2. An OMT without the third field is functionally equivalent to other trees. The third value in an OMT binds the first value to a record (in an IOMT) or a range to some "owner" (in an ROMT).

Alternately, a skip-list *and* a merkle tree are *together* functionally equivalent to an OMT. From this perspective, the main advantage of the OMT is that with a simple tweak to the merkle tree, the OMT realizes the advantages of ordering values (viz., ability to readily determine existence and non-existence of records, maximum/minimum values, etc.) without using an additional tree.

While there is very little algorithmic difference between an ROMT and an IOMT, there is a substantial difference in their functional utility. In this paper we illustrated the utility of an ROMT for registry and AS owner security kernels to maintain database of IP address ranges and the ownership of the range. In a BGP speaker the ROMT additionally enables the speaker to aggregate IP prefixes. The IOMT is used for a wide range of purposes like maintaining the RIDB, AS path trees (one for every path), and weight trees (one for every prefix).

The current approach to secure BGP is based on the Secure BGP [14] protocol proposed by Kent et. al. This approach employs public key certificates to authenticate communication between ASes (BGP updates) and delegation of AS numbers/IP prefixes. More specifically, a dual certificate system (supported in the back-end by a public key infrastructure (PKI)) is used where the one certificate binds the public key of the AS owner to the operating address space (IP prefix) and AS number, and a second certificate binds routers to an AS. Apart from such static certificates, dynamic certificates are also created by BGP speakers along with every update message. Specifically, such certificates created by every AS in the path seeks to assure the integrity of the AS path vector. Whenever a router receives an update message, it verifies the dual certificates to ascertain the validity of the message. In order to advertise the received message it extends the path by adding itself to the path and signing it (along with the nested signatures of the previous hops) with its own public key. To prevent deletion attacks a speaker in AS A sending an update message to a speaker in AS B also includes the next hop B in the signature.

While S-BGP approach is successful in its claims for identity verification (AS owner, routers) and update message integrity, it fails to provide any assurances for the overall operation of a sub-system in the protocol. For example, there are no assurances provided by the protocol guaranteeing that a router will indeed select the best path and that it will strictly abide by the policies and preferences prescribed by the AS owner. The security features of S-BGP protocol does not extend to aggregated prefixes as it is impractical to create static certificates to validate "ownership" of aggregated prefixes. This is a severe disadvantage of S-BGP as much of the advantages of CIDR stem from the ability to aggregate prefixes.

In the proposed approach the simple security kernel associated with BGP speakers ensure that the speakers can only advertise the best path, that all preferences and policies of the As owner will be strictly adhered to. More importantly, the assurances also extend to aggregated prefixes.

REFERENCES

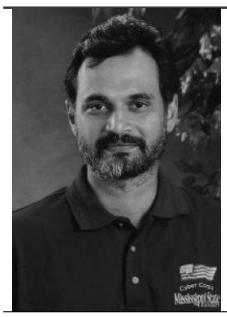
[1] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems*, vol. 10, pp. 265–310, 1992.

[2] E. R. Sparks, "A Security Assessment of Trusted Platform Modules Computer Science Technical Report," *Power*, pp. 1–29, 2007.

[3] P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine, "Authentic Third-party Data Publication," in *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions*. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2001, pp. 101–112. [Online]. Available: http://dl.acm.org/citation.cfm?id=646118.758638
[4] A. Buldas, P. Laud, and H. Lipmaa, "Accountable Certificate Management Using Undeniable Attestations," in *Proceedings of the 7th ACM conference on Computer and communications security*, ser. CCS '00. New York, NY, USA: ACM, 2000, pp. 9–17. [Online]. Available: http://doi.acm.org/10.1145/352600.352604
[5] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia, "Persistent Authenticated Dictionaries and Their Applications," in *Proceedings of the 4th International Conference on Information Security*, ser. ISC '01. London, UK, UK: Springer-Verlag, 2001, pp. 379–393. [Online]. Available: http://dl.acm.org/citation.cfm?id=648025.744371
[6] C. Martel, G. Nuckolls, M. Gertz, P. Devanbu, A. Kwong, and S. G. Stubblebine, "A General Model for Authentic Data Publication," *Algorithmica*, 2004.
[7] M. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing," in *DARPA Information Survivability Conference Exposition II, 2001. DISCEX '01. Proceedings*, vol. 2, 2001, pp. 68 –82 vol.2.
[8] R. C. Merkle, "Protocols for Public Key Cryptosystems," *Security and Privacy, IEEE Symposium on*, p. 122, 1980. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/SP.1980.10006
[9] Y. Rekhter and T. Li, "A border gateway protocol 4 (bgp-4)," 1995.
[10] Y. Rekhter and P. Gross, "Application of the border gateway protocol in the internet," 1995.
[11] W. A. Arbaugh, D. J. Farbert, and J. M. Smith, "A Secure and Reliable Bootstrap Architecture," in *IN PROCEEDINGS OF THE 1997 IEEE SYMPOSIUM ON SECURITY AND PRIVACY*. IEEE Computer Society, 1997, pp. 65–71.
[12] S. Bratus, E. Sparks, and S. W. Smith, "TOCTOU, Traps, and Trusted Computing," in *In Trust 08: Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies*, 2008, pp. 14–32.
[13] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated Data Structures for Graph and Geometric Searching," in *Proceedings of the 2003 RSA conference on The cryptographers' track*, ser. CT-RSA'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 295–313. [Online]. Available: http://dl.acm.org/citation.cfm?id=1767011.1767042
[14] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 4, pp. 582–592, 2000.

BIOGRAPHY OF AUTHORS

Somya D. Mohanty Dr. Somya D. Mohanty received his Masters degree in Computer Science from Florida State University and his doctorate from the department of Computer Science and Engineering at Mississippi State University. His doctoral research focuses on designing security kernels for distributed applications. Somya is currently working as a Data Scientist/Systems Architect on the Social Media Tracking and Analysis System (SMTAS) project with the Innovative Data Laboratory at the Social Science Research Center. He designs system architectures capable of handling Big Data and develops algorithms to gain insights from the data in real-time. He also contributes to the server architecture design with the use of dynamic scalable components capable of handling large data influx (Big Data). The work involved in SMTAS also includes researching current advances in social media and data mining technologies. Somyas other research interests include information/network security, cryptographic protocols, content analysis, machine learning and distributed storage architectures.



Ramkumar Mahalingam Dr. Mahalingam Ramkumar is an Associate Professor of Computer Science and Engineering at MSU. He received his Bachelors degree in Electrical Engineering from University of Madras, India, MS in Electrical Engineering from Indian Institute of Science, Bangalore, India, and PhD in Electrical Engineering from New Jersey Institute of Technology, Newark, NJ, in Jan 2000. He served as the Chief Technology Officer for a technology start-up in Newark between Feb 2000 to Sep 2002, and as a Research Assistant Professor in Polytechnic University, Brooklyn, NY from Oct 2002 to July 2003. His current research interests include trustworthy computing, applied cryptography, and network security. His has authored 2 books, 20 Journal articles/book chapters, and over 70 refereed conference publications.