An Efficient TCB for a Generic Content Distribution System

S.D. Mohanty, A. Velagapalli, M. Ramkumar
Department of Computer Science and Engineering
Mississippi State University, MS.

***Abstract*—We consider the security requirements for a broad class of content distribution systems where the content distribution infrastructure is required to strictly abide by access control policies prescribed by owners of content. We propose a security solution that identifies a minimal trusted computing base (TCB) for a content distribution infrastructure, and leverages the TCB to provide all desired assurances regarding the operation of the infrastructure. It is assumed that the contents and access control policies associated with contents are dynamic.**

I. Introduction

A content distribution system includes *publishers* who create content for consumption by *subscribers*, and a third party in the form of a *distribution infrastructure*, as publishers themselves may not possess the infrastructural capabilities required to distribute content. In any such content distribution system (CDS), publishers desire mechanisms to ensure that their content is made available only to a select set of subscribers, by prescribing an access control list (ACL) for the content. Subscribers desire mechanisms to ensure the integrity and authenticity of the content.

Irrespective of the nature of the specifics of the CDS, the users of the system - viz., publishers and subscribers - are expected to trust the infrastructural elements to preserve integrity of content, and adhere to the (content specific) ACL prescribed by the publisher. In practice, the "infrastructure" may be composed of possibly several agencies, computers and personnel. As malicious behavior by any infrastructural component may lead to violation of the desired assurances, and as it is impractical to rule out such behavior in complex systems, explicit mechanisms are required to assure the operation of such infrastructural elements.

A. Trusted Computing Base

The trusted computing base (TCB) [1] for a system is a small amount of hardware and/or software that need to be trusted in order to realize the desired assurances. More specifically, the assurances are guaranteed even if all elements outside the TCB misbehave.

The lower the complexity of the elements in the TCB, the lower is the ability to hide malicious/accidental functionality in the TCB components. Consequently, in the design of any security solution it is necessary to *lower the complexity of components in the TCB* to the extent feasible. The contribution of this paper is a broad security solution for assuring the operation of a content distribution system (CDS), and is motivated by the question "what is a minimal TCB for a CDS?"

In the proposed approach the TCB is a set of simple functions $\mathcal{F}()$ executed inside a trusted boundary - for example, by a trusted module $\mathbf{T}$. All desired assurances regarding operation of the CDS are guaranteed as long as the following (very reasonable) assumptions hold:

1) a pre-image resistant cryptographic hash function $h()$ exists.
2) the module $\mathbf{T}$ is read-proof and write-proof; in other words, the secrets protected by the module cannot be exposed, and the simple functionality $\mathcal{F}()$ of the module cannot be modified;

A multitude of proven hash functions (like SHA-1) exist that justify the first assumption. To justify the second assumption it is essential that the functionality $\mathcal{F}()$ is constrained to be simple enough to permit consummate verification. Towards this end we deliberately limit the computational and storage capabilities of module $\mathbf{T}$ required to execute $\mathcal{F}()$ - by constraining $\mathcal{F}()$ to be composed of simple sequences of logical and cryptographic hash $h()$ operations.

The contribution of this paper is an algorithmic description of a TCB functionality $\mathcal{F}()$ which can be leveraged to assure the operation of a broad class of content distribution systems.

The rest of this paper is organized as follows. In Section II we outline a generic model for a CDS and enumerate the desirable features and assurances. In Section III we provide a systemic overview of the proposed approach. In Section III-B we provide an overview of a simple data structure, an index ordered merkle tree (IOMT) which is used to succinctly represent all content handled by the CDS and the ACL for each content. Section IV provides a description of the operation of the system along with an algorithmic description of the TCB functionality $\mathcal{F}()$.

II. A Model for a Generic Content Distribution System

A content distribution system consists of a dynamic set of users $\mathcal{U} = \{u_1 \cdots u_n\}$ (who may be publishers, or subscribers, or both) and a dynamic set of content $\mathcal{C} = \{c_1 \cdots c_m\}$, where u_i is a unique identity of a user, and c_i is a unique label assigned to a content. Both sets $\mathcal{U}$ and $\mathcal{C}$ may be dynamic, and posses practically unlimited cardinality (unlimited n and m).

Associated with a content with label c_j are

1) a user of the system u_i identified as the publisher/owner of the content;

2) a content encryption secret s_j,

3) an access control list $\mathbf{A}_j$ (created by the owner), and;

4) a cryptographic hash γ_j of the encrypted content.

As long as a mechanism exists to securely deliver the content hash γ_j to a user, irrespective of the channel over which the actual content is delivered, the user receiving content c_j can verify the integrity of the content. For example, such encrypted content could be made available for download from a public repository or even distributed across several peer-to-peer clients. However, to gain clear access to the content, a user u requires the content encryption secret s_j. Only privileged users specified in the access control list (ACL) should be able to do so.

More generally, the ACL could assign various levels of privileges; for example, some users may be allowed only to access the content (more specifically, to receive secret s_j); some users with a higher privilege may be allowed to modify the content; some users with an even higher privilege may even be allowed to modify the ACL (in other words, the ACL may also be dynamic).

The infrastructural elements $\mathcal{I}$ in a CDS include mechanisms required to physically host content and the ACL associated with the content, accept requests from users, and deliver encrypted content and content decryption keys to privileged users.

A. *Desired Assurances*

Ideally, publishers should need to interact with the CDS infrastructure $\mathcal{I}$ only for uploading their content, or for modifying the ACL. Some of the specific desired assurances regarding the operation of the CDS are as follows:

1) $\mathcal{I}$ will not alter the content; more specifically, $\mathcal{I}$ will ensure that only users explicitly granted the permission (by the owner) to modify the content can do so.
2) $\mathcal{I}$ will not reveal content encryption secrets to unauthorized parties;
3) $\mathcal{I}$ will not modify the ACL; more specifically, $\mathcal{I}$ will ensure that only users explicitly granted the permission (by the owner) to modify the ACL can do so.
4) a user u authorized (as per the most recent ACL) to receive content will receive only most up-to-date version of the content;
5) when a user u requires access to content c_j, and if u *is* authorized access to the content, $\mathcal{I}$ will *not* refuse to provide access to the content.
6) when a user u requests access to content c_j, and if the content c_j does not exist, the user will learn nothing about the existence of other contents that have not been explicitly queried by the user; similarly, if u is *not* authorized access to the content, the user will learn nothing about other users who have access to the content.

Broadly, the first assurance is towards *authentication and integrity* of content. The second and third assurances are necessary to guarantee *privacy* of the content as intended by the creator. The fourth assurance addresses replay attacks - after a content has been modified, the older version may not be replayed by the CDS infrastructure; similarly, after an ACL has been modified, the old ACL should not be used to distribute content.

The fifth assurance is towards *authenticated denial* to prevent improper denial of service to a subscriber with a legitimate request. For this purpose, the infrastructure is expected to respond to *every* query. The response should either provide the requested content - or should contain a justification to convince the user that the requested content cannot be provided.

The sixth assurance is required to address some of the *undesirable side effects of providing authenticated denial*. In providing the proof of denial no information that was not explicitly queried should be provided[1].

III. Overview of Proposed Approach

In the proposed model, the untrusted infrastructure $\mathcal{I}$ has access to a trusted module $\mathbf{T}$ which serves as the TCB for $\mathcal{I}$. As every component of $\mathcal{I}$ is untrusted, the nature of the specific components of $\mathcal{I}$ (like personnel, computers and software) is irrelevant for our purposes. Some component of the $\mathcal{I}$ communicates with the module $\mathbf{T}$ using fixed interfaces exposed by the module. For example, the module $\mathbf{T}$ could be plugged into a computer in $\mathcal{I}$. Alternately, the module $\mathbf{T}$ could be housed in a secure location and connected over a (possibly untrusted) network to a computer in $\mathcal{I}$.

The module $\mathbf{T}$ is assumed to be capable of computing a shared secret with any user. Due to a wide variety of options that exist to establish such a shared secret, *in this paper we shall ignore* specific features in the module required to satisfy this requirement. We shall simply represent by K_i the secret common to a user u_i and the module $\mathbf{T}$. Such secrets are employed by users and the module $\mathbf{T}$ to authenticate requests and responses using message authentication codes (MAC), and for securely conveying content encryption secrets. The module is assumed to be capable of executing some simple functions $F_{adl}(), F_{inc}(), F_{upd}()$, $F_{cac}()$, and $F_{snd}()$ which are described algorithmically later in this paper.

In the proposed approach all desired assurances enumerated in Section II-A are guaranteed to the extent we can trust

[1]Ignoring such assurances in practical applications have sometimes resulted in attacks that undermine the utility of the protected system. For example, in DNSSEC [3], the security protocol for assuring the operation of a domain name system (DNS) server the unsolicited DNS records obtained from querying non existent records result in the undesirable "DNS walk" problem [4].

the integrity of the module. Specifically, that the module is read-proof implies that only a user u_j and the module have access to the secret K_j, and thus impersonation of messages is infeasible. That the module is write-proof implies that the functionality of the module is cannot be modified even by entities who have physical access to the module.

A. TCB Functions

In the proposed model TCB function $F_{inc}()$ is used for making a content available for distribution. The inputs to $F_{inc}()$ are various parameters (like content hash, encryption secret, ACL, etc) associated with the content c_j, and are authenticated by the owner o_i of the content for verification by $\mathbf{T}$ using a MAC computed using the secret K_i. After executing $F_{inc}()$ the module outputs an acknowledgment message for verification by o_i.

The TCB function $F_{upd}()$ is used to modify the content or modify the ACL associated with the content. Inputs of the function $F_{upd}()$ are authenticated by a user u_a who is (authorized to modify the content/ACL) using the secret K_a shared with the module. The output of the module is an authenticated acknowledgment. If the user is not authorized, the module will respond with an acknowledgement for a message indicating failure to carry out the request.

The input to $F_{snd}()$ is an authenticated request from a user u_q for some content c_j. Only if the user is authorized to receive the content will the module convey the content encryption secret s_j and content hash γ_j (to enable verification of content integrity) to user u_q. If the user is not authorized, the module will respond with an acknowledgement indicating inability to carry out the request.

In all exchanges between users and the module, $\mathcal{I}$ is an untrusted middle man. The middle man $\mathcal{I}$ is expected to faithfully perform some tasks in order to provide additional inputs $\mathbf{v}$ required for the TCB functions, and receive an authenticated acknowledgement from the module, which can then be delivered to the user. More specifically, if $\mathcal{I}$ does not execute such tasks faithfully, then $\mathcal{I}$ will not be able to obtain an authenticated response from the module to satisfy the user.

Unlike the three functions above, the inputs to $F_{adl}()$ and $F_{cac}()$ do not include authenticated requests from users. $F_{cac}()$ is employed by $\mathcal{I}$ to request the module to verify access control permission for a user, and generate a certificate vouching for the same. As this certificate is intended for verification by the same module (which issues the certificate) at a later time, the self-certificate is simply a MAC computed using a secret known only to the module $\mathbf{T}$.

Interface $F_{adl}()$ is used by $\mathcal{I}$ to request the module to insert or delete a leaf in an index ordered merkle tree (IOMT). The IOMT, which is treated in greater depth in the next section, is a simple extension of the better known (plain) merkle hash tree [2]. The main differences between a "plain" merkle tree and an IOMT are a) some additional rules to be observed in the IOMT for *inserting* and *deleting* leaves - to ensure uniqueness of indexes, and b) the ability to efficiently deal with any number of leaves - even if the number of leaves is not a power of 2.

In the proposed approach each leaf of the IOMT corresponds to a content. Associated with a set of m leaves (where m is the total number of content identifiers) are $2m-1$ hashes which are the "internal nodes" of the tree. The number m is assumed to be dynamic - m grows as content with new labels are introduced into the system (a leaf is inserted into the IOMT) for distribution and may reduce (an IOMT leaf is deleted) as their circulation is cut off by the owner (or an entity authorized by the owner). The IOMT is also used to efficiently represent the ACL associated with each content.

Untrusted $\mathcal{I}$ stores all m contents, $2m-1$ hashes, and some values associated with each content (leaf). Specifically, such values associated with a content c_j include a) content owner u_i, b) content hash γ_j, c) ACL $\mathbf{A}_j$ for the content, and d) an *encrypted* version of the content encryption secret s_j.

The module $\mathbf{T}$ stores only i) a single hash - the root ξ of the IOMT; and ii) a secret[2] S (known only to the module) used for encrypting the content encryption key.

Computationally, the module performs simple sequences of hash operations to execute functions $F_{adl}(), F_{inc}(), F_{upd}()$, $F_{cac}()$, and $F_{snd}()$, for which the module requires temporary scratch-pad memory for a mere $\mathbb{O}(\log_2 m)$ hashes (at most a few kB).

B. Index Ordered Merkle Tree

An index ordered merkle tree is a binary tree. A leaf of the IOMT is of the form (a, v_a, a') where a is an index, v_a is a value associated with the index, and a' is the next index. A leaf with index 0 is considered to be *empty*. A nonempty leaf (a, v_a, a') in the tree indicates that no leaf with an index that is covered by (a, a') exists in the tree. Specifically, an index c is *covered* by (a, a') if $cov(c, (a, a'))$ is true, where

```
$cov(c, (a, a'))\{$
RETURN $(a < c < a') \vee (c < a' < a) \vee (a' < a < c)$;
$\}$
```

A leaf $(a, v_a, a' = a)$ indicates that it is the *sole* leaf of the tree.

The IOMT employs two functions $H_L()$ and $H_N()$ - both of which are derived from a cryptographic hash function $h()$ (for example, SHA-1). The former takes an IOMT leaf as input and outputs a hash. The latter, takes two hashes as inputs and outputs a hash. The two functions can be expressed mathematically as follows.

$$v = H_L(a, v_a, a') = \begin{cases} h(a, v_a, a') & \text{if } a \neq 0 \\ 0 & \text{if } a = 0 \end{cases} \tag{1}$$

[2]In addition, the module may require to store one or more secrets that enable the module to compute pairwise secrets.

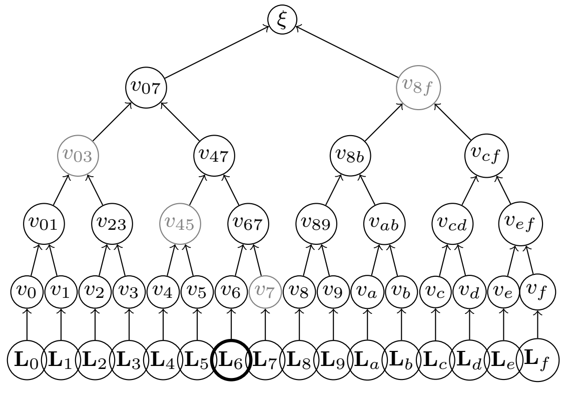
ξ
v_{07}
v_{8f}
v_{03}
v_{47}
v_{8b}
v_{cf}
v_{01}
v_{23}
v_{45}
v_{67}
v_{89}
v_{ab}
v_{cd}
v_{ef}
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_a v_b v_c v_d v_e v_f
$\mathbf{L}_0$ $\mathbf{L}_1$ $\mathbf{L}_2$ $\mathbf{L}_3$ $\mathbf{L}_4$ $\mathbf{L}_5$ $\mathbf{L}_6$ $\mathbf{L}_7$ $\mathbf{L}_8$ $\mathbf{L}_9$ $\mathbf{L}_a$ $\mathbf{L}_b$ $\mathbf{L}_c$ $\mathbf{L}_d$ $\mathbf{L}_e$ $\mathbf{L}_f$

Figure 1. An IOMT tree with 16 leaves. Nodes v_{67}, v_{47}, v_{07} and r are "parents" of v_6. v_{07} is the "immediate parent" of v_6. v_7 is the sibling of v_6, and v_{45}, v_{03} and v_{8f} are siblings of it's parents. The set of such siblings $\{v_7, v_{45}, v_{03}, v_{8f}\}$ is the set of nodes "complementary" to v_6. Similarly, the parents of v_{23} are v_{03}, v_{07} and ξ. The complementary set of v_{23} is $\{v_{01}, v_{45}, v_{8f}\}$.

$$p = H_V(u, v) = \begin{cases} u & \text{if } v = 0 \\ v & \text{if } u = 0 \\ h(u, v) & \text{if } u \neq 0, v \neq 0 \end{cases} \tag{2}$$

In Figure 1 the hashes $v_1 \cdots v_f$ at level 1 of the tree (just above the leaves $\mathbf{L}_1 \cdots \mathbf{L}_f$) are "leaf nodes" computed as $v_i = H_L(\mathbf{L}_i)$. The tree has $\lceil \log_2 m \rceil$ levels. Two adjacent nodes at a level i are hashed together using $H_N()$ to compute the immediate parent at level $i + 1$ (for example, v_{07} is the parent of v_{03} and v_{47} - or $v_{07} = H_N(v_{03}, v_{47})$. A tree with m leaves ($L = \log_2 m$ levels) has 2^{L-i+1} nodes at level i (m at level 1, $m/2$ at level 2, and so on and a single node (the root of the tree) at level L.

If all leaves are empty then all nodes at level 1 are zero (as $H_L()$ applied to an empty leaf yields 0). Consequently the root of the tree is zero. An IOMT with root $\xi = 0$ zero can be considered as a tree with *any* number of empty leaves. A root of an IOMT with a single leaf $\mathbf{L}$ is simply $H_L(\mathbf{L})$.

To see that only $2m - 1$ nodes will need to be stored even if m is *not* a power of 2, consider a scenario where only 14 leaves are non empty. If $\mathbf{L}_e$ and $\mathbf{L}_f$ are empty, then $v_e = v_f = v_{ef} = 0$ and $v_{cf} = v_{cd}$. Thus, the nodes v_e, v_f, v_{ef}, v_{cf} need not be stored. Similarly, if only $\mathbf{L}_f$ were empty, then $v_f = 0$ and $v_{ef} = v_e$ need not be stored.

Given any v at a level i, and the complementary set $\mathbf{v}$ with k nodes, a sequence of $H_N()$ operations map v to v's parent at level $k + i$. We shall denote by $p = f(v, \mathbf{v})$ the sequence of such operations. As a specific example (in the figure) corresponding to $v = v_{67}$, $\mathbf{v} = \{(v_{45}, 0), (v_{03}, 0), (v_{8f}, 1)\}$ includes 3 internal nodes and a bit associated with each node which indicates whether the internal node is to the left (bit set to 0) or right (bit set to 1) of $v = v_{67}$. The function $f()$ in this case executes $H_V()$ three times starting with v_{67} as $x_1 = H_V(v_{45}, v_{67}) = v_{47}$, $x_2 = H_V(v_{03}, x_1) = v_{07}$ and $x_3 = H_V(x_2, V_{8f}) = \xi$.

More specifically, if $\mathbf{v}$ is a set of L complementary hashes (along with L bits), then $\xi = f(v, \mathbf{v})$ maps a leaf node v to the root. Specifically, if provided a leaf $\mathbf{L}_i$ is considered as genuine only if $\xi = f(v_i, \mathbf{v}_i)$ where $v_i = H_L(\mathbf{L}_i)$. The implication of such genuine a leaf $\mathbf{L}_i = (a, v_a, a')$ is are i) a value v_a is associated with index a, and ii) no leaf exists for an index covered by (a, a').

1) Insertion and Deletion of Leaves: Only one leaf can exist in an IOMT for any index. Thus, a leaf with an index c can be inserted only if no leaf with index c exists - a situation which can be inferred by verifying a leaf which *covers* c. Specifically, to insert a leaf for an index c two leaves i) an empty leaf $(0, 0, 0)$ and ii) a leaf for some other index a - say (a, v_a, a') such that (a, a') covers c needs to be demonstrated to be consistent with root ξ. After insertion the two leaves will be modified to (c, v_c, a') and (a, v_a, c) respectively. Similarly, when a leaf (x, v_x, x') needs to be deleted, another leaf $(b, v_b, b' = x)$ needs to be demonstrated to be consistent with the root ξ. After deletion, the former leaf becomes $(0, 0, 0)$ and the latter becomes (b, v_b, x').

Except for the case of insertion of the first leaf or deletion of a sole leaf, to insert or delete a leaf, two leaves will need to be modified simultaneously - as for insertion of the first leaf the root $\xi = H_L(\mathbf{L})$ where $\mathbf{L} = (a, v_a, a)$ is the sole leaf; for deletion of a sole leaf the root is simply set to zero.

C. Access Control List

The access control list $\mathbf{A}_i$ can be seen as a list of two tuples of the form (o_i, a_i) where o_i is the identity of a user assigned privilege a_i. We shall assume that $a_i = 0$ implies that the user o_i is *not* granted access, and $a_i = 1$ to imply that the user o_i is granted access. We shall also assume an empty access control list to imply that the content can not be distributed.

In such a list $\mathbf{A}_j = \{(o_1, a_1), (o_2, a_2), \ldots, (o_k, a_k)\}$ associated with some content c_j, and ordered by ascending order of o (or $o_1 < o_2 < \cdots < o_k$), that a tuple for o_{i+1} follows o_i (in the list $\ldots, (o_i, a_i), (o_{i+1}, a_{i+1}), \ldots$) implies that no tuple exists for an identity that falls between o_i and o_{i+1}. In such a case it is more meaningful to see the ACL as a three tuple of the form $\{(o_1, a_1, o_2), (o_2, a_2, o_3), \ldots (o_{n-1}, a_{n-1}, o_n), (o_n, a_n, o_1)\}$.

For example, in a ACL

$$\mathbf{A} = \{(o_1, 1, o_2), (o_2, 0, o_3), (o_3, 1, o_4), (o_4, 1, o_1)\} \tag{3}$$

1) the implication of the first tuple $(o_1, 1, o_2)$ is that o_1 is explicitly granted access, and that all users $o_1 < x < o_2$ are denied access;

2) from the second tuple, o_2 is explicitly denied access; all users $o_2 < x < o_3$ are granted access;

3) o_3 is explicitly granted access; all users $o_3 < x < o_4$ are denied access;

4) o_4 is explicitly granted access; as $(o_1 < o_4)$, users $x > o_4$, and users $x < a_1$ are denied access;

In some scenarios the privileged users may have different types of privileges. For example, we shall assume that $a_i = 2$ implies that the user o_i is allowed to modify the content, and $a_i = 3$ implies that o_i is permitted to modify the content, and the ACL for the content. For example, if $\mathbf{A}_j = \{(o_1, 0, o_2), (o_2, 3, o_3), (o_3, 1, o_4), (o_4, 0, o_5), (o_5, 2, o_1)\}$,

1) o_1 is denied access; all users $o_1 < x < o_2$ are provided regular access;

2) o_2 is granted enhanced privilege to modify the ACL; all users $o_2 < x < o_3$ are denied access;

3) o_3 is provided regular access; all users $o_3 < x < o_4$ are denied access;

4) o_4 is denied access; all users $o_4 < x < o_5$ are granted regular access;

5) o_5 is explicitly granted privilege to modify the content; users $x > o_5$ and $x < a_1$ are denied access;

In the proposed approach each tuple in an ACL is seen as a leaf of an IOMT. The root α of such an IOMT is a succinct representation of the ACL. Specifically, given the value α, a set of hashes $\mathbf{v}$ and an IOMT leaf (u_r, a_r, u'_r) the module can verify the leaf against the root α and infer the access control restrictions associated with user $u_q = u_r$ or any user u_q covered by (u_r, u'_r).

D. IOMT for Storing Content Leaves

An IOMT leaf for storing values associated with a content c_j takes the form (c_j, v_j, c'_j) where c'_j is the next label and $v_j = h(u_i, \gamma_j, s^s_j, \alpha_j)$ is a one way function of the owner u_i, content hash γ_j, s^s_j (the encrypted version of content encryption secret s_j) and α_j (the succinct representation of the ACL associated with the content c_j). The root ξ of this IOMT changes whenever a content is introduced, or deleted, or if the ACL for a content is modified. The dynamic root ξ of this IOMT is maintained inside the module.

Given values $c_j, c'_j,\ u_i, \gamma_j, s^s_j, \alpha_j$ along with a set of hashes $\mathbf{v}$ the module $\mathbf{T}$ can verify the integrity of all values associated with the content c_j (by computing $v = H_L(c_j, v_j = h(u_i, \gamma_j, s^s_j, \alpha_j), c'_j)$, and then verifying that $f(v, \mathbf{v}) = \xi$.

A leaf with middle value zero, say $(c_j, 0, c'_j)$ is a "place-holder" for a content and can be used to reserve a content label. After content specific values are received for the content from the owner, such values are bound to the leaf by appropriately setting value v_j and updating the root ξ.

Unlike the IOMT for ACL which is prepared at one go by the owner or an authorized agent, various leaves of the content IOMT are provided by different content owners. Thus, for maintaining such an IOMT the module needs the ability to insert and delete IOMT leaves.

IV. TCB Functions

The resource limited module $\mathbf{T}$ securely stores a secret S, spontaneously generated inside the module, and the root ξ of an IOMT. It is assumed that the module can compute a secret K_i it shares with any user u_i.

A. Secret S

This secret S (spontaneously generated inside the module) is used by the module to encrypt content secrets entrusted to the module. Specifically, for a content c_j associated with a owner u_i, the content encryption secret s_j is conveyed by the owner u_i to the module as $s'_j = h(K_i, \mu_{ij}) \oplus s_j$, where $\mu_{ij} = h(c_j, \gamma_j, \alpha_j, s_j, K_i)$ is a message authentication code (MAC) used to securely convey the content related values to the module $\mathbf{T}$. The secret s_j is then re-encrypted by the module as $s^s_j = h(S, c_j, \gamma_j) \oplus s_j$ and handed back to untrusted $\mathbf{I}$ for storage.

The secret S is also used by the module to generate *self-certificates* for verification by itself at a later time. Specifically, using the function $F_{cac}()$ the module can be requested to issue a certificate of the form $\mu_s = h(u, a, \alpha, S)$ which states that "a user with identity u and access control permission a is *consistent* with access control digest α. The function $F_{cac}()$ can be described algorithmically as follows:

```
$F_{cac}((u_r, a_r, u'_r), \mathbf{v}, u_q)\{$
IF $(u_q = u_r)$ $a_q := a_r$;
ELSE IF($cov(u_q, (u_r, u'_r)))$ $a_q := (a_r = 0)?1 : 0$
ELSE RETURN;
$\alpha = f(H_L(u_r, a_r, u'_r), \mathbf{v})$;
RETURN $\mu_s := h(u_q, a_q, \alpha, S)$;
$\}$
```

The inputs to $F_{cac}()$ include a leaf of a ACL IOMT with root α along with the hashes $\mathbf{v}$ necessary to verify the leaf against α. The function $F_{cac}()$ will output a certificate only if $u_q = u_r$ or if u_q is covered by (u_r, u'_r). The values u_q, a_q, α and μ_s satisfying $\mu_s = h(u_q, a_q, \alpha, S)$ can be provided to the module at any later time to convince the module that "for a content with ACL digest α, a user u_q has access restriction a_q."

B. Addition and Deletion of IOMT Leaves

In general, to add or delete an IOMT leaf two leaves need to be updated. Two leaf hashes v_l and v_r can be simultaneously mapped to the root r by mapping the leaf hashes to the common parent, and then mapping the common parent to the root. Let v_p be lowest common parent of two leaf nodes v^l_p and v^r_p, and let $v_p = H_V(v^l_p, v^r_p)$ (v^l_p and v^r_p are the left and right child of v_p), $v^l_p = f(v_l, \mathbf{v}_l)$, $v^r_p = f(v_r, \mathbf{v}_r)$, and $r = f(v_p, \mathbf{v}_p)$. Now,

$$\xi = f(H_V(f(v_l, \mathbf{v}_l), f(v_r, \mathbf{v}_r)), \mathbf{v}_p) \qquad (4)$$

The interface $F_{adl}()$ can be used to insert a leaf for an index a (if a is covered by another leaf), or deleting a leaf with index a (if another leaf exists that points to a). Specifically, when a leaf is inserted the middle value is set to 0. Only leaves with middle value 0 can be deleted. Henceforth in this paper we shall refer to an IOMT leaf of the form $(a, 0, a')$ as a "place-holder."

The module functionality $F_{adl}()$ for inserting or deleting a place holder can be algorithmically described as follows:

```
$\xi = F_{adl}((l, v_l, l'), (r, v_r, r'), i, \mathbf{v}_l, \mathbf{v}_r, \mathbf{v}_p)\{$
IF $(l = 0) \wedge (r = 0)$//Insertion of first leaf
    $h_l := 0; h'_l := H_L(i, 0, i); h_r := 0; h'_r := 0;$ //
ELSE IF $(l = 0) \wedge cov(i, (r, r'))$
    $h_l := 0; h_r := H_L(r, v_r, r'); h'_l := H_L(i, 0, r'); h'_r := H_L(r, v_r, i);$
ELSE IF $(r = 0) \wedge cov(i, (l, l'))$
    $h_r := 0; h_l := H_L(l, v_l, l'); h'_r := H_L(i, 0, l'); h'_l := H_L(l, v_l, i);$
ELSE RETURN;
    $\xi_1 = f(H_V(f(h_l, \mathbf{v}_l), f(h_r, \mathbf{v}_r)), \mathbf{v}_p);$ //Root before insertion
    $\xi_2 = f(H_V(f(h'_l, \mathbf{v}_l), f(h'_r, \mathbf{v}_r)), \mathbf{v}_p);$ //Root after insertion
IF $(\xi = \xi_1)$ $\xi := \xi_2;$ // insert index $i$
IF $(\xi = \xi_2)$ $\xi := \xi_1;$ //delete index $i$
RETURN $\xi$;
}
```

$\mathcal{I}$ invokes this function whenever a new content is created or when distribution of an existing content has to be stopped, or when queried for an non existent content. Specifically,

1) if a new content with unique label c_j is made available for distribution, a place holder for index c_j is inserted to reserve a leaf for c_j (after this function $F_{upd}()$ will be used to bind the content related values to the middle value of the leaf).
2) if a query for a content c_J is made and no content exists for index c_j, a place holder for index c_j is inserted; soon after the query is answered the place-holder may be deleted.
3) The function $F_{uac}()$ (to update ACL) is employed to convert a leaf to a place-holder (by setting the middle value to 0) for halting the distribution of the content. The place holder can be deleted if required using $F_{adl}()$

To insert a place holder, two leaves - an empty leaf, and a covering leaf (j, v_j, j') (or $cov((j, j'), i)$ is TRUE) are provided as inputs to $F_{adl}()$. To compute the root from two leaves three sets of complementary hashes are provided to the module. The module computes the roots ξ_1 and ξ_2 - the roots before and after insertion respectively. If the current root ξ is either ξ_1 or ξ_2 it is reset to ξ_2 or ξ_1 respectively. Specifically, if the current root is ξ_1, and if the leaves provided satisfy the condition for inserting index i, by setting the root to ξ_2 a leaf with index i is inserted. On the other hand, for the same inputs, if the current root is ξ_2 then by setting the root to ξ_1 a leaf corresponding to index i is *deleted*. Thus, even while $F_{adl}()$ only verifies the pre-requisites for inserting a place-holder, $F_{adl}()$ can be used for both insertion and deletion of place holders.

C. Distribution of Content

The owner u_i of the content c_j performs the following steps to make the content available to users:

1) assign a unique label c_j to the content;

2) choose a random content encryption secret s_j and encrypt content;

3) compute hash γ_j of the encrypted content;

4) compute root α_j of an ACL IOMT $\mathbf{A}_j$;

5) submit to $\mathcal{I}$, i) the encrypted content, ii) access control list $\mathbf{A}_j$, iii) γ_j and iv) values μ_{ij}, s'_j where

$$\mu_{ij} = h(c_j, \gamma_j, \alpha_j, s_j, K_i), s'_j = h(K_i, \mu_{ij}) \oplus s_j, \tag{5}$$

and K_i is a secret shared between user u_i and the module $\mathbf{T}$; $\gamma'_j = 0$ implies that this is the first version of the content.

On receipt of such a message from user u_i, $\mathcal{I}$ performs the following steps:

1) reserve label c_j for user u_i; for this purpose $\mathcal{I}$ employs the interface $F_{adl}()$ exposed by the module $\mathbf{T}$. At the end of this process a place holder $(c_j, 0, c'_j)$ consistent with the IOMT root ξ will be available satisfying $f(H_L(c_j, 0, c'_j), \mathbf{v}_j) = \xi$.

2) evaluate α_j using the ACL;

3) employ module interface $F_{inc}()$ to supply values that include u_i, α_j, γ_j, s'_j, μ_{ij}, provided by the user and values c_j, c'_j and $\mathbf{v}_j$ associated withe the corresponding IOMT leaf. On completion of the execution of $F_{inc}()$ the module will outputs an acknowledgement MAC

$$\mu'_{ij} = h(ACK, \mu_{ij}, K_i). \tag{6}$$

The function $F_{inc}()$ can be described algorithmically as follows:

```
$F_{inc}(c_j, c'_j, \mathbf{v}_j, u_i, \gamma_j, \alpha_j, s'_j, \mu_{ij})\{$
IF $f(H_L(c_j, 0, c'_j), \mathbf{v}_j) \neq \xi)$ RETURN;
$s_j = h(K_i, \mu_{ij}) \oplus s'_j;$
IF $(h(c_j, \gamma_j, \alpha_j, s_j, K_i) \neq \mu_{ij})$ RETURN;
$s^s_j := s_j \oplus h(S, c_j, \gamma_j); v_j := h(u_i, \gamma_j, s^s_j, \alpha_j);$
$\xi := f(H_L(c_j, v_j, c'_j), \mathbf{v});$
RETURN $\xi, s^s_j, \mu'_{ij} = h(ACK, \mu_{ij}, K_i);$
}
```

4) store s^s_j, and make appropriate modifications to the parent nodes of the updated IOMT leaf to be consistent with the new root ξ;

5) send the acknowledgment MAC μ'_{ij} to the user u_i.

On receipt of the acknowledgment the owner u_i is assured (to the extent the owner can trust the module $\mathbf{T}$) that all the expectations of the owner with regards to distribution of the content will be met by $\mathcal{I}$. More specifically, this trust is based on the premise that i) it is infeasible for any entity except the module $\mathbf{T}$ and user u_i to compute the MAC, and that ii) the module functionality cannot modified.

D. Updating Content and/or ACL

To update the content c_j, a user u_a authorized to do so encrypts the modified content with a new key s_{ju} and computes the hash γ_{ju} of the updated content. The user then submits

$$\mu_{aj} = h(c_j, \gamma_j, \alpha_j, \gamma_{ju}, \alpha_{ju}, s_{ju}, K_a), s'_{ju} = h(K_a, \mu_{aj}) \oplus s_{ju},$$

to $\mathcal{I}$.

On receipt of the request to update content $\mathcal{I}$ employs interface $F_{cac}()$ to receive a certificate attesting the access control permission for u_a. Then, $\mathcal{I}$ employs interface $F_{upd}()$ shown below to update the IOMT leaf for c_j.

```
$F_{upd}(c_j, c'_j, u_i, \gamma_j, s^s_j, \alpha_j, \mathbf{v}_j, u_a, a_a, \gamma_{ju}, \alpha_{ju}, s'_{ju}, \mu_{aj}, \mu_s)${
IF $(\mu_s \neq h(u_a, a_a, \alpha_j, S))$ RETURN; //Invalid certificate
$s_{ju} = s'_{ju} \oplus h(K_a, \mu_{aj})$;
IF $(\mu_{aj} \neq h(c_j, \gamma_j, \alpha_j, \gamma_{ju}, \alpha_{ju}, s_{ju}, K_a))$ RETURN; //Invalid Request
$v_j := h(u_i, \gamma_j, s^s_j, \alpha_j)$;
IF $f(H_L(c_j, v_j, c'_j), \mathbf{v}_j) \neq \xi)$ RETURN;
IF $(a_a < 2) \vee ((a_a < 3) \wedge (\alpha_{ju} \neq \alpha_j))$; //user not authorized
  RETURN $\mu'_{aj} = h(ACK, \mu_{aj}, 0, K_a)$;
IF $(\alpha_{ju} = 0) v_{ju} := 0$;
ELSE   $s^s_{ju} := s_{ju} \oplus h(S, c_j, \gamma_{ju}); v_{ju} := h(u_i, \gamma_{ju}, s^s_{ju}, \alpha_{ju})$;
$\xi := f(H_L(c_j, v_{ju}, c'_j), \mathbf{v}_j)$;
RETURN $\xi, s^s_{ju}, \mu'_{aj} = h(ACK, \mu_{aj}, K_a)$;
}
```

If the user is not authorized, this function returns $\mu'_{aj} = h(ACK, \mu_{aj}, 0, K_a)$; if the user is authorized the function returns $\mu'_{aj} = h(ACK, \mu_{aj}, K_a)$ and the encrypted version s^s_{ju} of the new content encryption key s_{ju} for storage by $\mathcal{I}$.

When the user u_a receives the MAC μ'_{aj} the user u_a is convinced that the content has been modified and thus, form this point onwards, the old content hash γ_j cannot be replayed by $\mathcal{I}$.

E. Querying Content

Any user can send a query for any content. To query a content c_j a user u_q (who shares a a secret K_q with $\mathbf{T}$) computes a MAC

$$\mu_{qj} = h(c_j, \nu, K_q) \qquad (7)$$

where ν is a random nonce selected by the user. The user sends values u_q, c_j, ν and μ_{qj} to the DC.

If the queried content does not exist $calI$ inserts a place holder for c_j. If the content exists and the user u_q is authorized access, the ACL associated with the content will possess a tuple of the form $(u_q, a_q > 0, u'_q)$ or $(u_r, a_r = 0, u'_r)$ where (u_r, u'_r) covers u_q. On the other hand, if the user u_q is *not* authorized, the ACL will possess a tuple $(u_q, a_q = 0, u'_q)$ or $(u_r, a_r > 0, u'_r)$ where (u_r, u'_r) covers u_q. In either case, $\mathcal{I}$ can employ function $F_{cac}()$ to obtain a certificate $mu_s = h(u_q, a_q, \alpha_j, S)$ from the module.

This certificate, along with values c_j, ν and μ_{qj} sent by the user are provided to the module using the interface $F_{snd}()$, which can be described algorithmically as follows:

```
$F_{snd}(c_j, c'_j, u_i, \gamma_j, s^s_j, \alpha_j, \mathbf{v}, u_q, a_q, \mu_s, \nu, \mu_{qj})${
IF $(\mu_{qj} \neq h(c_j, \nu, K_q))$ RETURN;
$v_j := (u_i = 0)?\, 0 :\ h(u_i, \gamma_j, s^s_j, \alpha_j)$;
IF $f(H_L(c_j, v_j, c'_j), \mathbf{v}) \neq \xi)$ RETURN;
IF $(v_j = 0)$ RETURN $\mu'_{qj} := h(c_j, \nu, 0, 0, K_q)$; //Content does not exist
IF $(h(u_q, a_q, \alpha_j, S) \neq \mu_s)$ RETURN;
IF $(a_q > 0)$RETURN $\mu'_{qj} = h(c_j, \gamma_j, s_j, \nu, K_q), s'_j = h(K_q, \mu'_{qj}) \oplus s_j$;
ELSE RETURN $\mu'_{qj} = h(c_j, 0, 0, \nu, K_q), 0$; //No access
}
```

If the user is authorized access, the module returns $\mu'_{qj} = h(c_j, \gamma_j, s_j, \nu, K_q)$ and $s'_j = h(K_q, \mu'_{qj}) \oplus s_j$. When such values are conveyed to the user by $\mathcal{I}$, the user gains access to the content encryption secret $s_j = s'_j \oplus h(K_q, \mu'_{qj})$ and can verify the MAC μ'_{qj}. The user may now fetch the encrypted content from untrusted components of the DC, verify its integrity, and decrypt the content using the secret s_j.

If the user is not authorized, or if the queried content does not exist, the module outputs a MAC $\mu'_{qj} = h(c_j, \nu, K_q)$. When $\mathcal{I}$ relays μ'_{qj} values to the querier, the querier is assured (to the extent the module is trusted) that the content does not exist, or the user does not have access to the content.

V. Related Work and Conclusions

In this paper we presented a security solution for a generic content distribution system by identifying and leveraging a minimal trusted computing base for a content distribution infrastructure. The proposed solution caters for dynamic content associated with dynamic access control lists.

In the proposed approach no component of the content distribution infrastructure is trusted to realize the desired assurances. Only a simple hardware module $\mathbf{T}$ is trusted. Due to the generic nature of the proposed CDS, such a module (with fixed functionality) can be mass produced and employed for a wide range of content distribution systems.

In comparison, prior approaches using the TPM-TCG [12] platform involve verification of the content distribution servers for the state of operation. This results in unwarranted trust in several hardware components such as CPU, RAM, and any peripheral that has direct access to the RAM, etc. The problem is further amplified by the feasibility of TOCTOU [9] attack, which exploits the ability to modify code/data stored in RAM after it is loaded, but before it is executed. The TCG approach also depends on verification of every software module that was loaded on to the system, which involves verification and tracking of every version and update that a software module undergoes, rendering this approach far from practical.

The virtual counter approach by Sarmenta. Et. Al. [10], aims at addressing the issues with the TCG approach by enhancing the ability of TPMs to maintain a merkle tree used in conjunction with the monotonic counter of the TPM. The monotonic counter is used to generate virtual counters, which are then attached to each file and stored as leaves of the Merkle Tree. This approach has a limited scope in that it only tries to address replay attacks. Specifically, the goal is to ensure that once a content has been modified, the older version can not be replayed by the server. The approach in [10] does not support access control policies or support authenticated denial.

An attack against the scheme in [10] were demonstrated in [11]. The attack leverages the fact that there is nothing that prevents the untrusted server to bind multiple virtual counters to a file. Such an attack is no possible against the proposed approach as employing content identities for indexing leaves of an IOMT ensures that it is not possible to have multiple leaves corresponding to the same index. the untrusted server can generate multiple counters for the same content and then be able to replay an older version. The model also does not support authenticated denial for the content it does not possess.

Models for content distribution and security mechanisms for such models, have attracted substantial attention in the literature. Simple models that cater only for static content and ACL employ broadcast encryption [5], [6]. Specifically, in such systems the ACL associated with a content is a list of users/devices that are allowed access to the content, and is specified at the time the content is made available for distribution. The content is accompanied by a *message key block* consisting of various encryptions of the content encryption key such that at least one can be decrypted by a privileged device, while none can be decrypted by any revoked device.

More sophisticated models with dynamic content and ACL are generally considered under the umbrella of publish-subscribe systems [7],[8]. In such systems the infrastructure may consist of several servers, typically classified into servers at the provider end, consumer end, and core servers. While several security solutions have been proposed, they rely (to varying degrees) on the trustworthiness of servers in the infrastructure. Specifically in such systems the server is trusted to not replay old content, and not deny service to authorized users. One of the main motivations for the proposed approach stems from the lack of a suitable rationale for trusting complex servers employed by the CDS infrastructure.

Acknowledgement

This research was partly funded by the Department of Homeland Security (DHS)-sponsored Southeast Region Research Initiative (SERRI) at the Department of Energy's Oak Ridge National Laboratory.

References

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

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

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

[4] B. Laurie, G. Sisson, R. Arends, Nominet, D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," RFC 5155, March 2008.

[5] A. Fiat, M. Noar, "Broadcast Encryption," Lecture Notes in Computer Science, Advances in Cryptology, Springer-Verlag, **773**, pp 480–491, 1994.

[6] D. Noar, M. Noar, J. Lotspiech, "Revocation and Tracing Routines for Stateless Receivers," Lecture Notes in Computer Science, Advances in Cryptology, Springer-Verlag, **2139**, 2001.

[7] C. Wang, A. Carzaniga, "D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems," in *Proc.of Hawaii Intl. Conf. on System Sciences (HICSS)*, 2002.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," ACM Computing Surveys (CSUR), vol. 35, no. 2, pp. 114131, 2003.

[9] S. Bratus, E. Sparks, and S. W. Smith, "TOCTOU, Traps, and Trusted Computing," in *Trust 08: Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies*, pp. 14–32, 2008.

[10] L. F. G. Sarmenta, M. van Dijk, C. W. O'Donnell, J. Rhodes, and S. Devadas, "Virtual Monotonic Counters and Count-Limited Objects using a TPM without a Trusted OS," in *Proceedings of the first ACM workshop on Scalable trusted computing*, STC '06, (New York, NY, USA), pp. 27–42, ACM, 2006.

[11] S. D. Mohanty, M. Ramkumar, "Securing File Storage in an Untrusted Server Using a Minimal Trusted Computing Base," First International Conference on Cloud Computing and Services Science, Noordwijkerhout, The Netherlands, May 2011.

[12] "TCG Specification: Architecture Overview, Specification Revision 1.4," August 2007.