A trustworthy assurance-as-a-service architecture

Article · January 2015

DOI: 10.3233/978-1-61499-484-8-831

2 authors, including:

Mahalingam Ramkumar
Mississippi State University
124 PUBLICATIONS **1,032** CITATIONS

SEE PROFILE

A Trustworthy Assurance-as-a-Service Architecture

Mahalingam Ramkumar and Somya Mohanty

Department of Computer Science and Engineering
Mississippi State University, MS.

Abstract. Increasing complexity and inter-dependency of information systems (IS), and the lack of transparency regarding system components and policies, have rendered traditional security mechanisms (applied at different OSI levels) inadequate to provide convincing confidentiality-integrity-availability (CIA) assurances regarding any IS. We present an architecture for a generic, trustworthy assurance-as-a-service IS, which can actively monitor the integrity of any IS, and provide convincing system-specific CIA assurances to users of the IS. More importantly no component of the monitored IS itself is trusted in order to provide assurances regarding the monitored IS.

1 Introduction

An information system (IS) is a network of hardware and software used to create, collect, filter, process, and distribute data. The shrinking gap between the physical and the digital world portends a future where almost every crucial system we interact with in a day-to-day manner will be an IS operating over the Internet.

Behind our rapidly increasing dependence on ISes is an ugly truth — that there is simply *no tangible reason* as to why any one can trust the integrity of *any* IS. The reasons for our inability to make meaningful assertions regarding the integrity of any real-world IS are two fold: *complexity*, and *lack of transparency*. ISes are composed of numerous software/hardware components of unknown origin, possibly under the control of unknown personnel. It is impractical to rule out undesired functionality (either deliberately introduced malicious functionality or accidental bugs) in any system component, or malicious behavior/ incompetence in personnel able to influence the operation of such components. The lack of transparency — ignorance regarding the actual components of the system, or even broad policies that are actually enforced by the owners/controllers of the system — further exacerbates this problem.

Enhancing the utility of an IS demands the ability to provide consummate confidentiality-integrity-availability (CIA) assurances regarding the IS. The contribution of this paper is a broad architecture for a special IS — a Trustworthy Assurance-as-a-Service IS ($\mathrm{TA}^3\mathrm{S}$ IS)— that is intended to provide consummate CIA assurances to users regarding *other* ISes.

1.1 $\mathbf{TA^3S}$ IS

An obvious pre-requisite for the $\mathrm{TA}^3\mathrm{S}$ IS is that it should *not* suffer from the same issues that plague other ISes — complexity and lack of transparency. Three restrictions are imposed on the $\mathrm{TA}^3\mathrm{S}$ IS.

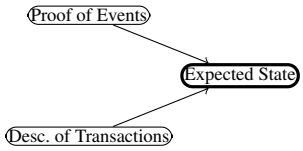
1. TA^3S IS is homogenous; it is composed of just one type of simple building blocks; we shall refer to such building blocks as TA^3S server modules (TSMs).
2. TSM functionality is *simple*, *fixed* and *open*.
3. No other assumptions (other that the integrity of TSMs) should be required for assuring the integrity of the TA^3S service.

TA^3S IS makes no assumptions about the integrity of any component of any IS monitored by the TA^3S IS. Thus, the integrity of reports from the TA^3S IS (regarding integrity of ISes monitored *by* the TA^3S IS) can be trusted to the extent the integrity of TSMs are assured. The restriction that TSMs have simple, fixed, and open functionality can go a long way towards the ability to perform consummate testing and certification of TSMs. TSMs are constrained to perform only simple sequences of cryptographic hash and logical operations, and demand very little memory.

Notwithstanding the deliberately introduced restrictions, the TA^3S IS is intended to be generic enough to provide comprehensive CIA assurances for *any* IS. At one end of the spectrum, an IS monitored by TA^3S could be as large as the entire Domain Name System (DNS) [1], or the entire inter AS (autonomous system) Internet routing infrastructure, based on the Border Gateway Protocol (BGP) [2]. At the middle of the spectrum, a monitored IS may be intended for an organization (for example, Banner system for a university). At the other end of the spectrum, the monitored IS could be as small as a DHCP server serving a local network. In the rest of this paper we outline some of the important functional components of TSMs, and strategies used by TA^3S IS to leverage the trust in TSMs to provide convincing CIA assurances.

2 Overview of TA^3S

Any IS can be seen as a set of (possibly distributed) databases. Assuring the integrity of an IS can then be seen as the process of ensuring that only well-formed database transactions can be performed.



In general, a transaction is triggered by an *event*. Thus, proof of occurrence of the actual sequence of events (starting from the time when all databases were empty), along with a description of all event-specific transactions is sufficient to determine the current *expected state* of any IS.

A comprehensive list of various possible events, and event-specific transactions, can be made available by the designer of the system, and be reviewed for correctness. If there is also a mechanism to confirm the actual occurrence of all events, there is no need to rely on any component of the IS itself to determine current expected state of the IS. Users[1] of an IS can be seen as entities who make specific queries to the IS. *If* users can unambiguously determine the expected state of an IS, any response from the IS that is inconsistent with the expected state can be construed as proof of lack of integrity of the IS.

As it is impractical for users to verify proofs of occurrence of *every* event over the entire lifetime of the IS, one possible approach is to provide the ISes themselves the

[1] A "user" can also be another IS.

option to continually submit proof of events to a trusted third party (TTP). The TTP tracks the expected state of the IS, and responds to queries from users regarding the IS state. Users interacting with the IS can now verify the integrity of any response from the IS by comparing it with a report of the expected state of IS from the TTP. To the extent the users are convinced of the integrity of the TTP, the users can be assured of the integrity of the monitored IS. The $\mathrm{TA}^3\mathrm{S}$ IS serves as the TTP. All that an IS (that desires to provide consummate CIA assurances to it's users) needs to do is to continually provide convincing proof of occurrence of events to one or more TSMs of the $\mathrm{TA}^3\mathrm{S}$ IS.

TSMs are assumed to be thoroughly verified, and certified for integrity. Every TSM is issued a unique identity and possesses secrets that enable a) any two TSMs to establish a secure channel, and b) permit users to receive authenticated integrity reports (regarding monitored ISes) from TSMs. It is assumed that secrets of TSMs are well-protected (to ensure that TSMs can not be impersonated), and TSM functionality can not be modified.

2.1 Clark-Wilson Model

$\mathrm{TA}^3\mathrm{S}$ has some similarities with the Clark-Wilson (CW) model for system integrity [3],[4]. In the CW model, all data to which the model is applied (data whose integrity need to be guaranteed) are identified and labeled. These are *constrained data items* (CDI). The CW model defines integrity verification procedures (IVP) which take CDIs as input, and outputs a binary value indicating if the CDIs represent the system in a valid/invalid state. CDIs can be modified only by well formed transaction procedures (TP).

The correctness of IVPs and TPs are certified by a "security officer" for the system. Well formed TPs are guaranteed to take the system into a correct state — *if* the system was at a correct state *before* the execution of the TP. That a system is in a correct state can initially be verified by executing an IVP. From this time onwards, if only a sequence of well-formed TPs are executed, by induction, the system is guaranteed to remain in a correct state.

The CW model also recognizes unconstrained data items (UDI) that represent new inputs fed into the system — the external triggers for modifications to the CDIs. TPs that handle UDIs will need to be certified for their ability to recognize and reject invalid UDIs.

$\mathrm{TA}^3\mathrm{S}$ **vs CW:** In the $\mathrm{TA}^3\mathrm{S}$ model, similar to the CW model, only those values that have a bearing on the desired assurances are considered as CDIs. While the CW model does not place any constraint on the nature of CDIs, the $\mathrm{TA}^3\mathrm{S}$ model constrains CDIs to be represented as a special data structure — $\mathrm{TA}^3\mathrm{S}$ CDIs are databases represented as leaves of an index ordered Merkle tree (IOMT). The UDI's in CW model are similar to events in the $\mathrm{TA}^3\mathrm{S}$ model. The $\mathrm{TA}^3\mathrm{S}$ model does not need a separate IVP as an all empty database can be seen as a valid starting point for most systems.

The most important difference between the two models is in the restrictions imposed on TPs. The only restriction on TPs imposed by CW model is through specification of CW-triples of the form (user, TP, CDIs) that restrict which user process is allowed to execute a TP, and restrict which CDIs can be modified by a TP. The CW model

simply *assumes* the run-time integrity of TPs, which is unrealistic, especially if TPs are executed on a general purpose computer. The $\mathrm{TA}^3\mathrm{S}$ model, on the other hand, proactively aims to simplify representation of TPs to permit TPs to be executed within the trustworthy resource limited boundary of TSMs.

$\mathrm{TA}^3\mathrm{S}$ **Databases:** The databases maintained by real words ISes will in general be substantially different from $\mathrm{TA}^3\mathrm{S}$ databases that capture the CDIs corresponding to the IS. For example, an actual domain name system (DNS) database may be comprised of DNS records, indexed by name and type. Corresponding to each index there may be multiple (say n) DNS records. In a corresponding $\mathrm{TA}^3\mathrm{S}$ database the index may be a one-way function of name and type (for example, name and type hashed together). Corresponding to each index is a cumulative hash of all records for the same name and type.

As another example, consider a cloud storage service where users desire assurances of integrity and freshness regarding files stored in the cloud. Specifically, assume that users desire that no one except users explicitly authorized by the owner of the file can modify the file, and that the service will always provide only the freshest version of the file. For this purpose, the $\mathrm{TA}^3\mathrm{S}$ database can consist of records indexed by unique file indexes, indicating i) a file hash corresponding to the most recent version of each file; ii) a counter indicating the highest version number; and iii) an access control list (ACL) for the file. As the TSM is trusted to not permit modifications to file hashes or version numbers by users not the in the ACL for the file, comparing the file hash supplied by an TSM to the actual file fetched from the cloud provides assurances of integrity; the version number provided by the service can similarly be compared with the version number provided by the TSM to be assured of freshness.

The databases used by the actual IS (a real life cloud storage system) can be significantly different. The IS may use a different indexing scheme for files — one that makes it possible to readily identify the directory the file is located in, the owner of the file, frequency of access, etc. The IS may also maintain a database of different computers belonging to a user, and a list of folders that need to be synced in each computer. The service may also maintain a database with a record for each user indicating user quota, an the actual space utilized by the user. The reason that the $\mathrm{TA}^3\mathrm{S}$ database does not care about all other databases maintained by the IS is simply due to the fact that assurances regarding file quota or correctness of sync have not been defined as desired assurances. *If* the IS desires to provide more consummate assurances to it's users, the scope of the $\mathrm{TA}^3\mathrm{S}$ databases corresponding to the IS may be enhanced.

It is important to note that the only link between real world databases of an IS and the $\mathrm{TA}^3\mathrm{S}$ database for the IS is that *both receive the same external triggers (events).* While the changes triggered by an event in the actual IS can be influenced by innumerable (and possibly unknown) factors, the changes triggered by the event in the $\mathrm{TA}^3\mathrm{S}$ database is clearly defined by designer of the system, and enforced by a TSM. For example, an event involving a request from a user u who is *not* authorized to modify a file f will be considered as an "event to be ignored" by the $\mathrm{TA}^3\mathrm{S}$ TSM. If the actual IS honors such an event, it's databases will no longer be consistent with the $\mathrm{TA}^3\mathrm{S}$ database. Consequently, the IS and will no longer be able to demonstrate it's integrity to users for all future transactions involving file f.

3 Salient Features of TA^{3}S IS

Two of the salient features of TA^{3}S are 1) the use of a simple-yet-flexible authenticated data structure, an index ordered Merkle tree (IOMT) [5] -[7], to represent any TA^{3}S database; and 2) interpreting any IS as a network of any number of TA^{3}S databases, possibly of different types, where the integrity of each database is tracked by a dedicated TSM.

3.1 Index Ordered Merkle Tree

An IOMT is a binary hash tree [8]. A tree with N leaves has $2N-1$ nodes distributed over $L = \log_2 N$ levels. A single node at level L of the tree is the *root* of the tree. An entity storing only the root of the tree (a single hash) can verify the integrity of any internal node, or any leaf, by performing a sequence of not more than $L = \log_2 N$ hash operations. Thus, all nodes and leaves of the tree can be stored in any convenient (possibly insecure) location.

Binary Hash Tree Functions: For any binary tree, verifying the integrity of a node x against an ancestor node y, say k levels higher than x, will require a verification object (VO) consisting of k internal nodes — one from the same level as x, and one from each level between the levels of x and y. If the VO (a vector of k hashes) is $\mathbf{x}$ then we can define a simple function $\boxed{\tilde{y} = f_{bt}(x, \mathbf{x})}$ where, *if* $y = \tilde{y}$ it can be inferred that x is indeed a node in a sub-tree with root y. The only way this inference can be wrong is if it is possible to break the second pre-image resistance property of the cryptographic hash function used for constructing the tree (which can safely be assumed to be impractical for any good cryptographic hash function).

If there is a legitimate need to modify node x (to say x'), the same function $f_{bt}()$, along with the verified VO $\mathbf{x}$, can also be used to update the root y of the sub-tree as $\boxed{\text{IF } y = f_{bt}(x, \mathbf{x}) \text{ THEN } y' = f_{bt}(x', \mathbf{x})}$.

IOMT Leaves: Two of the main features of the IOMT that distinguish it from other binary trees are i) the leaves of an IOMT have a rigid structure, that constrains all leaves of an IOMT to form a circular linked list; and ii) IOMT supports simple algorithms for dynamic insertion and deletion of leaves. An IOMT leaf is a three-tuple of the form `(index, next_index, value)`, where `index` is the index of a record and `value` is a one-way function (for example, cryptographic hash) of the contents of the record for the index. It is also possible that `index` is the index of a database, in which case `value` is the root of nested IOMT.

As the leaves form a linked list, the existence of a leaf $(5, 10, x)$ simultaneously conveys non-existence of records with indexes that lie between 5 and 10. Due to the *circular* nature of the linked list, the existence of a record like $(100, 2, x)$ (where `next_index` is less than `index`) implies non-availability of records with indexes greater than 100, or less than 2 (alternately, 100 is the largest index, and 2 is the least index). The first two values in a leaf (say index $= a$ and next_index $= a'$) is said to enclose a value b (or (a, a') encloses b) if $a < b < a'$ or $a' \leq a < b$ or $b < a' \leq a$.

The requirement that the leaves always form a circular linked list is enforced whenever a leaf is inserted or deleted. A leaf for index an b can only be inserted if a leaf (a, a', x_a) enclosing b already exists in the tree. After insertion of the new index b the

next_index for a becomes b, and the next_index for b becomes a'. Similarly to remove a leaf for index c with next_index c' a leaf like (d, c, x_d) (with next_index as c) is modified as $(d, c, x_d) \rightarrow (d, c', x_d)$).

TSMs of the $\mathrm{TA}^3\mathrm{S}$ IS will have the ability to verify the integrity of any record in an $\mathrm{TA}^3\mathrm{S}$ database corresponding to a monitored IS, and/or confirm absence of records with specific indexes in the $\mathrm{TA}^3\mathrm{S}$ database. To gain this ability they merely have to store a single hash inside the secure confines of the TSM, and execute function $f_{bt}()$ — which will involve $L = \log_2 N$ hash operations for a database with N records (for example, only 30 hash operations for a database with a billion records).

3.2 $\mathrm{TA}^3\mathrm{S}$ Network

From the perspective of $\mathrm{TA}^3\mathrm{S}$, an IS S (monitored by $\mathrm{TA}^3\mathrm{S}$) is simply a network of $\mathrm{TA}^3\mathrm{S}$ databases. Let $\mathbf{D}_1^s \cdots \mathbf{D}_n^s$ represent $\mathrm{TA}^3\mathrm{S}$ databases corresponding to IS S. Each database is tracked by a dedicated TSM. Corresponding to an IS S is thus a $\mathrm{TA}^3\mathrm{S}$ *network* (TN) of TSMs $a_1^s \cdots a_n^s$ that verify occurrence of events, and track databases $\mathbf{D}_1^s \cdots \mathbf{D}_1^s$ (respectively).

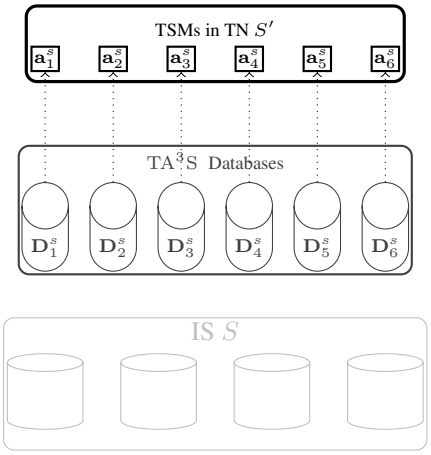


Fig. 1. $\mathbf{D}_1^s \cdots \mathbf{D}_6^s$ are $\mathrm{TA}^3\mathrm{S}$ databases for IS S. $\mathbf{a}_1^s \cdots \mathbf{a}_6^s$ are dedicated TSMs that track the $\mathrm{TA}^3\mathrm{S}$ databases $\mathbf{D}_1^s \cdots \mathbf{D}_6^s$ (respectively). The specific contents of $\mathrm{TA}^3\mathrm{S}$ databases for an IS S will depend on the desired assurances for S. TSMs tracking $\mathrm{TA}^3\mathrm{S}$ databases of IS S are members of "TSM network" (TN) S'.

A TN (a set of TSMs) corresponding to an IS itself has a unique identity. Let S' be the identity of the TN corresponding to IS S. Every TSM that belongs to the TN S' is a *member* of the TN S'. According to TN member a_i^s (a TSM), which tracks the $\mathrm{TA}^3\mathrm{S}$ database $\mathbf{D}_i^s$, a single hash (IOMT root) ξ_i serves a concise summary of the database $\mathbf{D}_i^s$. More specifically, any record that can be demonstrated to be consistent with ξ_i, is interpreted by a_i^s as a record in the $\mathrm{TA}^3\mathrm{S}$ database $\mathbf{D}_i^s$. The physical location of the $\mathrm{TA}^3\mathrm{S}$ database is irrelevant from a security standpoint as the entity that maintains the $\mathrm{TA}^3\mathrm{S}$ database and the IOMT for the $\mathrm{TA}^3\mathrm{S}$ database need not be trusted. If the IS so chooses, the IS itself can maintain the $\mathrm{TA}^3\mathrm{S}$ database. If the IS does not desire to perform the additional tasks necessary to prove the integrity of it's $\mathrm{TA}^3\mathrm{S}$ databases to TSMs, the assurance service provider $\mathrm{TA}^3\mathrm{S}$ could offer this additional service to IS S.

Dynamic TNs: While there are any number of ways to break down a complex database into a network of smaller databases, a useful approach can be based on the type of each component database. For example, the DNS is an IS that can be seen as a network of interdependent databases corresponding to DNS registry database, database of DNS registrars, zone owner database, zone server database, etc. Corresponding to different types of IS database there may be different types of $\mathrm{TA}^3\mathrm{S}$ databases. Thus, members of a TN can be seen as having different roles, depending on the type of $\mathrm{TA}^3\mathrm{S}$ database monitored by an TN member. In a TN for monitoring the DNS IS, TN members can be seen as having different roles like DNS registry TSM, zone owner TSM, DNS server TSM, etc.

For practical large scale ISes the number of databases may also be dynamic. For example, in the DNS system, new DNS servers will have to be added, new zones will be created, etc. Correspondingly, new members will have to be added to the TN. To facilitate additions / deletions of TN members, each TN possesses a special TSM identified as the *creator* of the TN.

TN Identity: When operating in an TN S', an TN member a_i^s recognizes it's role. According to TN member a_i^s there exist "provable events" that necessitate modification to one or more records of $\mathrm{TA}^3\mathrm{S}$ database $\mathbf{D}_i^s$. Any number of such events, and transactions corresponding to such events may be specified by the designer of the TN. According to the TN member, the verification procedure for an event, and the procedural representation for the $\mathrm{TA}^3\mathrm{S}$ database transactions triggered by the event are specified as "blobs" bound to a leaf of a *static* IOMT – the *root of which is the TN identity* S'. In other words, according to TSM a_i^s

1. the static TN identity S' is a commitment to all permitted $\mathrm{TA}^3\mathrm{S}$ database transactions for an IS S.
2. the dynamic root ξ_i is a commitment to the current state of the $\mathrm{TA}^3\mathrm{S}$ database $\mathbf{D}_i^s$.

The identity of the creator TSM is also explicitly specified in the static IOMT. Thus, even if two different ISes X and Y have the same exact set of rules (for example, identical accounting systems deployed in different organizations) the TN identities X' and Y' will be different as they will have different TN creator identities.

Events: From the perspective of both the monitored IS and the $\mathrm{TA}^3\mathrm{S}$ IS changes to the databases are triggered by events. From the perspective of TSMs, an event can be a) a message from another TSM, or b) a message from an authorized (by the designer) external entity; or c) a spontaneous event triggered merely by the current state of the $\mathrm{TA}^3\mathrm{S}$ database. Occurrence of the event may require modification of one or more records, and possibly creation of a TN message (which could then trigger an event in another TN member).

TN messages convey a type, a value corresponding to the type, and the current time according to the sender. As such messages are authenticated using a secret shared between the sending and the receiving TSM, TN messages implicitly convey the sender and the receiver. While TN messages can not be impersonated (as it is assumed that secrets protected by TSMs can not be exposed), it is possible that untrusted entities who relay messages between TSMs may simply ignore the messages. To address such possibilities, TSMs expect every TN message to be acknowledged by the receiving TN.

More specifically, when a message μ_{ab} is created by TSM a for delivery to b, TSM a adds a reminder (to the $\mathrm{TA}^3\mathrm{S}$ database) of an outstanding acknowledgment from b. Only after the message is actually delivered to TSM b, and TSM b has successfully processed the message, will an authenticated acknowledgement be sent from b to a. As long as any acknowledgement is outstanding TSM a will not consider the monitored database to be in an acceptable state.

3.3 Mandatory and Discretionary Components

As $\mathrm{TA}^3\mathrm{S}$ IS is constructed entirely as a network of homogenous TSMs, a specification of TSM functionality is indirectly a specification for the $\mathrm{TA}^3\mathrm{S}$ IS. The broad functional components of an TSMs are

1. IOMT functions
2. Authentication and TN Messaging functions
3. TN management functions; and
4. Transaction functions

IOMT functions enable a TSM to verify the integrity of any record in the $\mathrm{TA}^3\mathrm{S}$ database or verify the absence of a record against a dynamic root ξ stored inside the TSM. In addition they also enable a TSM to verify the integrity of any transaction rule, or verify the absence of a rule against the static identity of the TN in which the TSM is operating as a member.

Authentication and TN messaging functions enable submission of proof of events in the form of authenticated TN messages. TN management functions enable dynamic TN memberships. One important requirement is to ensure that within a TN, every member is issued a unique identity. Note that if two TSMs are issued the same identity (and thus oversee the same $\mathrm{TA}^3\mathrm{S}$ database), it is possible that only subsets of events are supplied to each of them. To ensure that no member identity is issued to two different TSMs the TN creator can maintain an IOMT indexed by TN member identity.

All functions discussed above can be seen as mandatory components [9]. The discretionary IS specific components are simply transaction rules that identify how a message of a specific type leads to modification of a specific record in the $\mathrm{TA}^3\mathrm{S}$ database, and possibly the creation of a new message. Other discretionary components specified by the designer of the IS can include the structure of the IOMT (especially when the database is represented by nested IOMTs), format for records for different databases, restrictions on transactions (for example, by specifying roles of TN members that can engage in the transaction.

4 Discussions and Conclusions

In this paper we presented a broad architecture for an IS that is capable of providing convincing assurances regarding other ISes. More importantly, no component in the monitored IS needs to be trusted. Some of the obvious concerns regarding the viability of $\mathrm{TA}^3\mathrm{S}$ IS capable of monitoring any IS are a) is the $\mathrm{TA}^3\mathrm{S}$ approach scalable, especially given the resource limitations of TSMs? and b) is it possible for an approach

intended for assuring integrity of $\mathrm{TA}^3\mathrm{S}$ databases to also provide *confidentiality* and *availability* assurances?

Scalability: Issues that affect the scalability of $\mathrm{TA}^3\mathrm{S}$ IS are two fold: a) the size of $\mathrm{TA}^3\mathrm{S}$ databases, and b) the rate at which events that modify $\mathrm{TA}^3\mathrm{S}$ databases. The size if the database is less of a concern as the complexity of functions that process events increase only logarithmically with database size. However, the frequency of events is likely to increase linearly with the size of the database. For example, a $\mathrm{TA}^3\mathrm{S}$ database for a cloud storage system may consist of one record for every file. While a thousand fold increase (say from a million to a billion) will only increase the complexity of each procedure by a factor $30/20$, the frequency of events (number of updates for files per second) is more likely to increase by a factor 1000.

What is perhaps a more challenging issue is the *transaction rate*. Resource limited TSM may not be able to hundreds of thousands of transactions per minute (which may be the case for several real world systems). However, this can be addressed by carefully splitting $\mathrm{TA}^3\mathrm{S}$ databases into multiple databases, each tracked by a different TSM. As an example, the $\mathrm{TA}^3\mathrm{S}$ database for a cloud storage system can be split into multiple parallel databases, for different ranges of file indexes. A similar approach can also be used for say name servers for `.com` TLD in the domain name system.

Availability: From a security perspective, availability of a service, among other things, implies that the service is not incorrectly denied. For example, a DNS server should not be able to simply claim that a queried record does not exist when it actually does [10]. A cloud storage system should not be able to incorrectly deny service claiming that the file does not exist or that the user does not have access to the file [7]. Specifically, the service should be able to convincingly demonstrate that the queried object does not exist, or that the user does not have the necessary access permission for the object.

As the IS itself is untrusted, the non availability of the queried object or the lack of sufficient access permission should be verifiable by a TSM. This is made possible by the fact that the IOMT is a linked list. The TSM can verify the existence of a record like $(5, 10, x)$ to infer non-existence of records with indexes that lie between 5 and 10. That a TSM trusted by the user attests to the non existence of queried objects (by verifying a leaf that does exist) implies the TSM can simply assure the querier that the "request can not be entertained" without the need to provide additional information regarding objects that were not queried explicitly. Such a feature also has the additional advantage of eliminating data mining attacks that can be accomplished by random querying. For example, in the DNSSEC protocol the DNS server is required to respond to queries for non existent records by demonstrating a NSEC record [10] which attesting the names of two consecutive records that do exist (which enclose the queried name).

Confidentiality: As a TSM assures the integrity of an IS, such assurance messages will need to be cryptographically authenticated by TSMs. Obviously, secrets employed by TSMs for this purpose should be read-proof [11] – [12] to prevent impersonation of TSMs. In the proposed approach, users of a system are capable of establishing a shared secret with TSMs. This secret, which is used for authentication of assurance messages from TSMs, can also be used for securely conveying sensitive and private user data (for example, password, credit card number etc.) to TSMs. The TSM for the

system is trusted to not release such information, or release such information only in strict accordance with pre-specified rules. For example, in a cloud storage system the file encryption secrets could be handed over to TSMs [7]. TSMs are trusted to ensure that the secret is conveyed only to authorized users in the ACL for the file.

Our ongoing work focuses on arriving at a more detailed specification of TSM functionality, modeling ISes as CDI databases, and investigating transaction procedures for a wide range of ISes.

References

1. P. V. Mockapetris, "Domain names - concepts and facilities," RFC Editor, 1987
2. Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC Editor, 1995.
3. D.D.Clark, D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," in Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy (SP'87), May 1987, Oakland, CA; IEEE Press, pp. 184193.
4. X. C. Ge, F. Polack, R. Laleau, "Secure databases: An analysis of Clark-Wilson model in a database environment," Proceeding of Advanced Information Systems Engineering, 2004, pp 234-247.
5. V. Thotakura, M. Ramkumar, "Minimal TCB For MANET Nodes," 6th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2010), Niagara Falls, ON, Canada, September 2010.
6. A. Velagapalli, S. Mohanty, M. Ramkumar,"An Efficient TCB for a Generic Data Dissemination System," International Conference on Communications in China: Communications Theory and Security (CTS), ICCC12-CTS, 2012.
7. 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.
8. R.C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," Advances in Cryptology CRYPTO '87. Lecture Notes in Computer Science 293. 1987.
9. Trusted Computer System Evaluation Criteria. United States Department of Defense, December 1985, DoD Standard 5200.28-STD.
10. S. Weiler, J. Ihren, "RFC 4470: Minimally Covering NSEC Records and DNSSEC On-line Signing," April 2006.
11. S. W. Smith, "Trusted Computing Platforms: Design and Applications," Springer, New York, 2005.
12. M. Ramkumar, "Trustworthy Computing Under Resource Constraints With the DOWN Policy," IEEE Transactions on Secure and Dependable Computing, pp 49-61, Vol 5, No 1, Jan-Mar 2008.