A Structure Preserving Approach for Securing XML Documents

Invited Paper

Mohamed Nabeel
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907
Email: nabeel@cs.purdue.edu

Elisa Bertino
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907
Email: bertinol@cs.purdue.edu

Abstract—With the widespread adoption of XML as the message format to disseminate content over distributed systems including Web Services and Publish-Subscribe systems, different methods have been proposed for securing messages. We focus on a subset of such systems where incremental updates are disseminated. The goal of this paper is to develop an approach for disseminating only the updated or accessible portions of XML content while assuring confidentiality and integrity at message level. While sending only the updates greatly reduces the bandwidth requirements, it introduces the challenge of assuring security efficiently for partial messages disseminated to intermediaries and clients. We propose a novel localized encoding scheme based on conventional cryptographic functions to enforce security for confidentiality and content integrity at the granularity of XML node level. We also address structural integrity with respect to the complete XML document to which clients have access. Our solution takes every possible measure to minimize indirect information leakage by making the rest of the structure of XML documents to which intermediaries and clients do not have access oblivious. The experimental results show that our scheme is superior to conventional techniques of securing XML documents when the percentage of update with respect to original documents is low.

I. INTRODUCTION

The problem of securing the XML content transmitted in distributed systems with third-party architectures for data publishing has been widely investigated [1]–[3]. Today these architectures receive growing attention due to their ability to handle large number of users and vast amounts of data. XML, being an interoperable content representation, has become a de facto standard for representing content in such systems. Most of the solutions proposed view the content to be distributed as a whole, making it difficult to incrementally publish as and when portions of published content change. Our goal in this paper is to fill this gap while providing a platform for efficient, scalable and secure dissemination. The task is further complicated by the fact that different user groups may have different access levels to published XML documents. Our approach can be used to publish only the updates to an XML document, while achieving security properties in a holistic manner. We extend the idea of querying encrypted data [2]–[4] in our work.

In a typical distributed system, data originate from document source(s) and go through zero or more intermediaries (content routers) before reaching final destinations (clients). Further, clients may either only have access to, or want to receive selectively, portions of the disseminated XML documents. While the three-tier architecture provides scalability, we can maximize the entropy and minimize bandwidth requirements by employing a partial message dissemination mechanism where we publish a minimum amount of data about the rest of the message or document. Bandwidth efficiency and scalability would be of little significance without assuring the key security requirements. Therefore, any useful dissemination approach should also maintain the following two security requirements.

1) Integrity to provide users with enough data to verify that any partial XML document sent is not tampered by unauthorized parties.
2) Confidentiality to guarantee that users are given access to only the portions of the XML documents to which they are authorized and that the existence of data to which users don’t have access is hidden, thus minimizing indirect information leakage.

Our focus on securing XML content in distributed systems is not new. There has been substantial amount of research on ensuring confidentiality and integrity by encrypting different portions of the XML document with different symmetric keys [2], [5], and by introducing encrypted post-order numbers [3].

While some underlying techniques used in our approach have been used in different areas of computer science, our approach differs from earlier efforts in three key ways. First, we optimize our framework to secure incrementally updated XML documents. Second, we propose and implement algorithms to encode and encrypt XML documents that are faster than existing techniques such as post-order number based encoding [3]. We augment our claims with a series of experimental results. Third, we make a clear distinction between sibling ordering and hierarchical ordering in XML documents, which is described in the next section, to further improve the efficiency of our scheme in terms of processing.
time and bandwidth utilization.

The reminder of this paper is organized as follows. The next section introduces the underlying key concepts behind our approach. Section 3 presents the proposed annotation and encoding scheme, and their runtime complexities. Section 4 describes how the two key security requirements, confidentiality and integrity are enforced. Section 5 provides a set of experimental results to prove the viability of our approach. Section 6 discusses about related work and their drawbacks. Section 7 concludes the paper and looks at future research directions.

II. BASIC CONCEPTS

In this section, we start with different XML entity orderings that are of interest to us and present how we exploit them for our advantage. We then discuss about the granularity of data for which we should be able to reason about security. Finally, we show how these concepts fit into our approach.

We follow the same terminology of the XML specification [6] and the DOM specifications [7] to refer to physical and logical structures in the XML document respectively. The document is composed of units called physical entities or logical nodes. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are specializations of nodes.

In the context of this paper, we can identify two major ordering of entities in an XML document.

1) Hierarchical ordering
2) Sibling ordering

Hierarchical ordering determines the parent-child relationship of entities in an XML document. Sibling ordering determines the order of children with respect to their parent. We cannot think of an application where hierarchical ordering is not significant. XML Infoset [8] stipulates that sibling ordering should be preserved at all times. However, we observe that this decision depends on the application we are dealing with. If the application is document-centric, such as when transforming an XML document into a different textual format using XSLT, we do need to respect the sibling ordering to preserve the correctness. However, if the application is data-centric, such as when mapping an XML document to a relational database or a class, sibling ordering becomes insignificant.

We take advantage of the above observations by providing two levels of granularity in our proposed annotation scheme, where we annotate each element node in the XML document to enforce and verify security requirements, in order to gain performance benefits for data-centric applications. The first level enforces structural integrity on the hierarchical ordering and the second on the sibling ordering. The two levels together match the ordering rules in the XML Infoset.

We need to be very clear of the smallest unit of data that can be published. With incremental updates, we publish only the changes to an XML document. With partial accesses, we expose only the portions to which clients have access. An XML document consists of element as well as non-element entities. In order to manage updates or split along document boundaries, we first divide the entities into element entities which correspond to element information items in the XML infoset and non-element entities which correspond to rest of the items in the infoset. The key idea behind encoding, which will become clear in the next section, is to convert an XML document into a one with only element entities while carrying all the significant information in the original document. We reason about security requirements at the level of these encoded element entities. Therefore, the smallest change that we publish boils down to an encoded element entity. Thus, whenever an element information item or a non-element information item associated with an encoded element changes, we publish that encoded element.

The other side of the coin is the smallest unit of data that a client can subscribe to. A similar argument can be employed to show that it also has the same granularity of an encoded element entity.

We can identify two types of security; TLS (Transport Level Security) such as HTTPS and IPSec and MLS (Message Level Security). TLS provides only point-to-point security and, therefore, is not sufficient to protect data which need to go through intermediaries such as content routers which may or may not be trusted. We need to provide end-to-end security for the kind of distributed systems we are dealing with. Hence, our approach is to secure every message disseminated through the network providing MLS.

Our approach as a whole is to secure each and every message, partial or full, disseminated through the network without having to trust intermediaries or the infrastructure. Further, our approach allows one to send minimal amount of data to recipients without leaking information to which they do not have access or without propagating duplicate information along with the updates.

III. ANNOTATION AND ENCODING SCHEME

First we look at the motivation and the intuition behind our annotation and encoding scheme and then move on to look at how we implement. Any annotation and encoding scheme used should possess the capability to handle the following two scenarios efficiently.

1) Reasoning about the security of partial messages published with minimum indirect information leakage.
2) Allowing clients to access portions of messages which they are authorized to access while preventing them from accessing data they are not allowed to access.

The common denominator of the above two scenarios is that the scheme proposed should be able to process each element entity as independently as possible. In other words, clients should be able to verify security properties of the partial messages received and intermediaries should be able to respond to client queries for data respecting access control policies.

We need to annotate each element node to reason about structural integrity, assuring the relative order of nodes in an XML document when viewed as a DOM tree. Annotating
an XML document tree with various additional information for security as well as non-security purposes is not new [3], [9]. These annotation schemes use tree traversal order (post-order, in-order and pre-order) requiring each node in the XML document to be traversed sequentially. Therefore, they essentially have a time complexity of $O(n)$ where $n$ is the number of nodes in the XML document. Instead of having one single global scheme, we propose a two-level ordering scheme which has the following benefits over its predecessors.

1) The annotation scheme becomes local in that each branch of the XML DOM tree can be traversed in parallel. We can achieve a much better time complexity of $O(h)$ where $h$ is the height of the DOM tree. For a nearly balanced DOM tree the complexity becomes almost $O(\log n)$ where $n$ is the number of nodes in the XML document.

2) The two level scheme corresponds to hierarchical and sibling ordering, allowing us to enforce structural integrity at two different granularities depending on the level of structural integrity that applications demand.

3) The local annotation scheme offers more flexibility to incrementally annotate as and when the structure of the original XML document is changed.

Figure 1 shows one possible way of traversing a DOM tree in parallel. Starting from the root of the DOM tree, we independently process each sub-tree. The algorithm recursively branches into further sub-trees at any given node if it has at least two element child nodes. We use this parallel tree traversal scheme in most of the algorithms we implemented to traverse the DOM tree.

Fig. 1. Concurrent Traversal of XML DOM Tree

A. Annotation Scheme

A key point of our scheme is that the two orderings, hierarchical and sibling, are orthogonal, thus allowing us to exploit the parallelism in the annotation and other related algorithms we propose. In order to reason about hierarchical ordering, we assign an identifier to each element in the XML document which unambiguously identifies the parent-child relationship: if $x$ is an element with the identifier $ID_x$ and $y$ is a child element of $x$ with the identifier $ID_y$, $y$ keeps track of the identifier pair $(ID_y, ID_x)$. It should be noted that identifiers themselves are not required to be unique. The elements at the same rank (they have the same height from the root) in the DOM tree, can have the same identifier, but the elements at different ranks must have different identifiers.

**Lemma 1:** By annotating each element with its identifier and its parent identifier, the hierarchical ordering of any subtree of the XML DOM tree can be deterministically verified, and hence the hierarchical structural integrity can be verified.

**Proof:** Proof by induction.

The only requirement is that we should be able to unambiguously identify parent-child relationships. One very secure way of realizing this is to use XPath [10] based identifiers with a collision-resistant hash function. We may not be able to use XPath directly as it may leak ordering information when there are more than one sibling element with the same name. While there are many ways to overcome this drawback, we may use the following simple solution.

Given that element $x$ is the parent of element $y$ and $XP_x$ is the XPath of $x$, the identifier of $y$ is defined as $h(XP_x | | element name of y)$ where $h$ is the collision-resistant hash function and $||$ is the concatenation operation.

The second level of the ordering, the sibling ordering, is achieved by annotating each sibling with a secure random real number maintaining the following condition.

**Condition:** Given that elements $x$ and $y$ are siblings and $x$ is to the left of $y$, $seq_x < seq_y$ where $seq_x$ and $seq_y$ are secure random numbers assigned to $x$ and $y$ respectively.

We maintain this condition by annotating each sibling element from left to right with a strictly increasing number.

The use of secure random real numbers for sibling ordering over integers has the following added benefits even though it requires more computation.

1) It makes inferring information about siblings to which clients do not have access difficult, if not impossible. For example, it is computationally difficult to determine how many nodes there are to the right of the rightmost sibling accessible to a client.

2) It provides the flexibility to add/remove siblings later without requiring re-annotation. Theoretically we can have an infinite number of real numbers between any two real numbers. However, a higher level of security can be achieved by manipulating the parameters of the random number generator based on the rough estimation of how much change to the original structure is anticipated.

If the sibling ordering is insignificant for the applications under consideration, we do not need to annotate elements with sibling ordering to verify structural integrity. Hierarchical integrity implies structural integrity in such cases.
B. Encoding Scheme

The key idea behind the encoding scheme is to convert XML document consisting of elements and non-elements into another XML document consisting of only elements without losing any significant information from the original document.

Definition 1 (Encoding function) Let E be any element and EE denote the encoded element corresponding to the original element E.

We encode the following information to EE obtained from the original element E and its parent and non-element children.

\[
EE.tagname = E.id \\
EE.order = E.order \\
\]

where || denotes the concatenation operation with a delimiter, E.tagname the tagname of element E, E.id the annotated identifier, E.parentId the annotated identifier of its parent, E.order the annotated sequence number, E.attr[1] the set of attribute name-value pairs associated with E, and E.child[1] the set of id-content pairs of non-element child nodes of E. For sibling ordering insignificant applications, we only need to encode the content of E.child[1] nodes leaving their identifiers and do not need to add E.order.

The encoded XML document has the following two key properties.

1) The XML document consists only of elements.
2) The number of elements in the encoded document is lower than the total number of nodes in the original document.

The above definition shows only the data that are encoded before enforcing any security requirement except structural integrity. We add additional data to the above encoded node in the next section where we discuss the two key security requirements laid down at the beginning.

IV. Enforcing and Verifying Security Requirements

In this section, we describe how we enforce and verify the integrity and confidentiality on the above encoded XML document. We then combine everything together and show the structure of a fully encoded XML element.

A. Integrity

We enforce integrity on the content as well as the structure of the XML document. The idea is to associate a hash value with each element to ensure the authenticity of that element and its non-element children (content integrity). Any conventional collision-resistant one-way hash function [11] can be employed for this purpose. The following equation illustrates it in precise terms.

\[
EE.signed = h(EE.attrs||EE.content)
\]

where || denotes the concatenation operation with a delimiter, h the hash function which produces EE.signed and all the other notations have the same meaning as in Definition 1.

The hash value computed is similar to the one computed with the Merkle hash function [12] in that we concatenate the element and its tag name to compute the hash value. However, our approach differs significantly from it in the following three ways.

1) Our signature scheme is not recursive as in the Merkle approach; we sign each element independently. As a consequence, a small change to a node in the XML document requires the re-computation of Merkle hash whereas our approach re-computes only the hash value of the element to which the node belongs.
2) Our scheme allows to verify each element in the XML document independently assuring minimum indirect information leakage about the portions of the document to which clients do not have access.
3) Merkle approach requires the XML tree to be signed from bottom-up whereas our approach is flexible to perform localized signing allowing us to exploit the parallelism.

We rely on standard canonical XML specifications [13, 14] to serialize XML documents before signing and verifying the content of it. Everything required to verify the authenticity of any received element is self-contained within the element. When the document source wants to send an update, it generates hash values for only those elements which are updated while keeping the rest unmodified.

We have already shown how to encode information to enforce structural integrity, which is further divided into hierarchical integrity and sibling integrity. E.id and E.parentId in Definition 1 are used to enforce hierarchical integrity. We use E.id to locate the element node in the DOM tree and compare E.parentId with the E.id of the parent element of E to verify the hierarchical integrity of E. According to Lemma 1, by verifying each element, we implicitly verify the hierarchical integrity of the complete XML document.

E.order in Definition 1 is used to enforce sibling integrity, an optional integrity level for data-centric applications, but a necessary integrity level for document-centric applications. To assure that the sibling ordering is preserved, we compare the E.order values of E’s left and right siblings, if exist, in the DOM tree to see if they follow the condition laid down in the Annotation Scheme section.

B. Confidentiality

We need to satisfy two aspects to make sure that the confidentiality is preserved at all times.

1) Using an access control model (Author-X [15] for example) to protect XML documents from unauthorized accesses.
2) Encrypting the content of each element node so that unauthorized parties cannot decipher the content.

Access control models for XML have widely been investigated [15]–[17]. Credential based access control models presented in [15], [16] are a good example of providing different access granularity levels along with a way to specify access control policies based on the document structure as well as content. Depending on the size of the distributed system, either the document source(s) or the intermediaries may authorize users. In the latter case, intermediaries need to be equipped with the access control policies and credentials to make authorization decisions.

In our model, the content of each element is encrypted independently from other elements in the XML document. Further, we prune away elements from the XML DOM tree according to the access control policies before sending the update to users. Therefore, we can use the same asymmetric key pair or the same symmetric key to encrypt the whole document alleviating us from cumbersome key management issues. We can use a key shared between the document source and clients, and a randomly generated key to encrypt the attribute name-value pairs, the content and the signed hash value of each element. The following equation illustrates it precisely.

\[
EE._{encrypted} = \text{key}_s(\text{key}_r) \mid| \\
\text{key}_c(\text{EE}.\text{attrs})|\text{EE}.\text{content}|\text{EE}.\text{signed}
\]

where \(\mid|\) denotes the concatenation operation with a delimiter, \(EE._{encrypted}\) encrypted data of element \(E\), \(\text{key}_s\) is the shared key, \(\text{key}_r\) is the randomly generated key and the other terms have the same meaning as described previously.

To summarize the complete encoding scheme, each encoded element \(EE\) with the tag name \(EE._{tag}\) has the attributes order (provided that we want to ensure sibling ordering for structural integrity), parentid, signed and encrypted with the values \(EE._{order}\), \(EE._{parentId}\), \(EE._{signed}\) and \(EE._{encrypted}\) respectively. Signed and encrypted data in each element is usually base64 [18] encoded at the document source end and decoded at client ends.

V. Reference Implementation and Experimental Results

The goal of the implementation is to investigate the viability of the proposed approach and compare it with existing approaches both structure preserving, such as the method based on post-order numbers, and non-preserving, such as the method based the W3C encryption and digital signature.

A. Implementation Details

In order to make the implementation less complicated and incremental processing possible, we follow the well-known technique of layer based approach to secure XML documents. A source XML document is signed and encrypted in three passes as shown below.

1) The XML document is annotated with structural identifiers which we use to verify structural integrity.

2) The DOM tree is encoded into a new DOM tree consisting only of elements. Each element encodes details about all the non-element children nodes.

3) The content of each element is independently signed and then encrypted in the same pass.

Once the document is encoded in the second pass, we repeatedly use the same DOM tree to partially or fully sign and encrypt the XML document based on which elements are updated. The ability to sign and encrypt any portion independently is a key requirement of systems where incremental updates are frequent. We provide this capability at the cost of manipulating each element separately.

Similarly, an encrypted XML document is decrypted and verified in two passes as shown below.

1) Each element is decrypted and its integrity is verified.

2) Each decrypted element is decoded in order to obtain the original DOM tree.

Similar to the forward passes, the decryption and verification can also be carried on any portion of the XML document independently which is also a key requirement of above mentioned systems. Each of these passes have one aspect in common; we can localize the processing to exploit the parallelism hidden inside the DOM tree. All our algorithms executed on the encoded tree follow a similar pattern. The following simple pseudo code provides a high level view of how each pass works. The visitor pattern [19] is a good way of separating different algorithms for encrypting, signing, verifying and decrypting from the parallel document traversal.

Visitble is an interface that has the visit() method. Each element also implements this interface. The following code is common to all passes. The idea is to have a thread pool initiated at the beginning and allocate a new thread from the pool to each independent branch in the XML DOM tree. The spawning of new threads can be further controlled by defining a threshold corresponding to the number of branches traversed before delegating the task of processing a new branch to another thread.

```java
DocumentTree implements Visitable {
    ... 
    accept(Visitor visitor) |
    visitor.visit(this) 
    foreach child of visitor { 
        if (first-child) { 
            visitor.accept(child) 
        } else { 
            Spawn a new thread 
            visitor.accept(child) 
        } 
    } 
}
```

We wrote a concrete Visitor class for each pass. Calling the accept() method of DocumentTree with an instance of this class will apply the algorithm implemented here on all elements in parallel.
Traverser implements Visitor {
    visit(Element element) {
        execute algorithms specific to the pass
    }
}

We use a separate bottom-up parsing algorithm to annotate XML documents and form encoded XML documents.

B. Experimental Results

In this section we present a summary of the experimental results gathered from our reference implementation and we compare and contrast the results with other approaches where applicable. For all the experiments, we assume the DOM tree corresponding to each synthesized XML document is already loaded into the memory. We performed our experiments on a 4 CPU, shared, multi-user Linux server and the development platform was JDK 1.5. We used the open source Apache XML Security library [20] as the reference implementation of W3C XML digital signature and encryption specifications [21], [22]. The library relies on the open source Bouncy Castle Java security provider [23] for encryption.

Figure 2 compares the time taken to annotate XML documents of different sizes using a global annotation scheme such as using tree-traversal order and a local annotation scheme that we use. As can be clearly seen from figure 2, our implementation with the parallel processing algorithm is, in general, more than 2 to 3 times faster than conventional tree-traversal based implementations. It is our opinion that we can get still better results with dedicated servers having multiple processors running the algorithm. We use the same underlying technique with the algorithms to encode, decode, sign, verify, encrypt and decrypt in order to improve the performance.

Figure 3 compares the time taken to partially signing a XML document with our scheme to signing with the W3C scheme.

We updated randomly selected portion of an XML document with over 1500 elements. As it can be seen from Figure 3, the time taken to sign the updated portions is comparable to W3C signature scheme. Our scheme gives better results when the number of element updated at a given time is less than 2% the total number of elements. We can achieve similar results for other passes in our layered approach.

As expected, our scheme takes much more time to sign complete XML documents than W3C scheme. However, the key point here is that in our scheme we expect to sign the complete document only once, at the system initialization stage or whenever a new document is added to the system. Subsequently, only the updated elements are signed and encrypted which is not possible with the W3C scheme where one needs to sign and encrypt the whole document even for the smallest possible change that can happen. Another well-known fact in any practical system is that adding bandwidth to the network is more costly than adding processing power to scale a system. This is especially true when a large number of messages are exchanged at a high rate. The high processing time is compensated not only by the low bandwidth utilization, but also by the preservation of the XML structure which allows one to query and update any portion of the XML document independently. Since each element is self-sufficient in that it contains enough information to verify the security requirements, our scheme minimizes the indirect information leakage even in the form of encrypted or signed data. None of the above mentioned properties are achieved by both W3C signature [21] and encryption schemes [22].

Figure 4 shows how much time each of the three processing steps, encoding, signing and encrypting takes for XML documents of different sizes.

With conventional digital signature and encryption algorithms, our experimental results indicate that the bulk of the time in securing an XML document is spent on signing the content of each element. By using faster cryptographic hash
functions we can improve the performance of our scheme.

One possible reason why XML infoset [8] stipulates that sibling ordering should be preserved is to make the life easier for serial validation tools and avoid performance penalties. However, the experimental results show that by making sibling ordering optional in our approach, we can not only achieve better performance for both enforcing and verifying structural integrity, but also reduce the size of the encoded messages. The performance improves since we omit all conditions and comparisons related to sibling ordering and the size reduces since we no longer need to encode sequencing information related to the sibling ordering into the encoded message.

VI. RELATED WORK

Our work is related to XML security and secure dissemination of XML content in distributed systems. We look at some of the key research work done along these lines.

Bertino and Ferrari [24], and separately Miklau and Suciu [5] propose a technique to assure confidentiality in published XML data using cryptographic techniques. The key idea is to partially encrypt different portions of the XML document with different keys meeting all policy requirements. Their attempt solves the problem of keeping different views of the same XML document for different user groups. However, their scheme does not prevent indirect information leakage as users have receive other encrypted portions of the document. Further, key management becomes an issue when a large user base with different policy settings has to access the same document. Also, such a technique is not suitable for systems producing incremental updates as it does not preserve the structure of XML documents and there is no way to encrypt only the updated sections. Opyrchal and Prakash [25] also investigate the problem of providing confidentiality in content-based publish subscribe systems while proposing techniques for aforementioned key management problem.

The idea of using encrypted XML documents in publish-subscribe systems is first introduced in [1] by Carminati et al. to assure authenticity and completeness of the XML documents published in a pull based approach and later extending the same idea but a completely new approach was proposed by Kundu and Bertino [3]. In the latter work, they have introduced the notion of post-order numbers not only to provide security for the XML documents published but also to route messages to subscribers. The proposed structure-based routing works well in a system where there are only a few large XML documents and the subscription to routers is well managed to group overlapping subscriptions into content routers.

All of the above approaches are designed to work with complete XML documents and do not address how to ensure integrity and confidentiality in a holistic manner in systems where XML documents are incrementally published.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented a practical approach to secure XML documents which can be used to, but not limited to, securely publish only the incremental updates or provide access to partial content.

Based on the notion of hierarchical and sibling ordering of elements in XML documents, we have proposed and implemented an efficient encoding scheme creating a sound foundation to secure XML documents. Based on this encoding scheme, we showed how to enforce the key security requirements, integrity and confidentiality while preserving the structure of XML documents. A key point in our approach is that we enforce and verify security at node level without requiring the knowledge of other nodes in the XML document. We also introduced the notion of two-level structural integrity which allows us to boost the performance and reduce the message size of data-centric applications where we do not need to reason about sibling ordering of XML elements to verify the structural integrity.

The experimental results show that our techniques and algorithms on enforcing security are practically viable especially when enforcing security on incrementally updated XML documents.

Due to the potential and its applicability in real life applications, we plan to extend the platform laid in this paper to support concurrent updates from many sources to the same XML document. We also plan to research into ways of still further improving the performance especially on signing documents where the bulk of the time is spent. Another area we would be keen to look into is how to combine message compression with our technique to further reduce bandwidth with minimal performance penalty.

ACKNOWLEDGMENT

The work reported in this paper is part of the project "Systematic Control and Management of Data Integrity, Quality and Provenance for Command and Control Applications" partially funded by the USA Air Force Office of Sponsored Research (Grant n. FA9550-07-1-0041)
REFERENCES


