Fine-Grained and Scalable Message Protection in Sensitive Organizations

Joon S. Park and Ganesh Devarajan
The Laboratory for Applied Information Security Technology (LAIST)
School of Information Studies
Syracuse University
Syracuse, New York, USA
{jspark@syr.edu, ganesh.devarajan@gmail.com}

***Abstract*— Today electronic messaging services are indispensable to individuals and organizations. If a message contains sensitive information, the integrity and confidentiality of the contents created by individual users should be maintained in an effective manner. Therefore, there is an urgent need for new mechanisms to support the requirement. In this paper we focus on the message protection in the organization-to-organization service, while many other researchers have worked on the person-to-person service. Our approaches can provide effective integrity verification, tracking mechanisms, and confidentiality at the user-level in a scalable manner with fine granularity. Furthermore, we use a set of session keys for each message to provide more secure communications, maintaining scalable key management by employing a key hierarchy. We describe the mechanisms of our approaches and show their feasibility by describing a prototype system that we developed.**

I. INTRODUCTION

When data is transferred between different sensitive divisions, domains, or even organizations, reliable control must be taken to maintain the integrity and confidentiality of the messages and files that were passed. In particular, if security levels of users are different, information must not flow from the high level to the low level [1]–[4]. In such cases we should verify the security levels of the users and the data being transferred. To validate data being transferred between different organizations, we enforce devices such as Guards, which are also known as Boundary Controllers, or Cross Domain Solutions (CDS). Any critical information that is to be transferred from Organization A to Organization B has to pass through the Guard, which checks, filters, and sanitizes data between the organizations based on policies. The Guard executes pre-defined security policies that are based on the trust level, guidelines, rules, directives, and instructions. Basically, when we interlink two organizations of different security levels, we should prevent sensitive data from being transmitted to unauthorized organizations or persons. The Guard also maintains logs for all requests and responses that are passed across organizations.

Considering message source and destination, generally, there are two different kinds of services: person-to-person and organization-to-organization. The former is just like ordinary e-mail service, where each sender selects the receiving person. The latter is for organization-level communications (e.g., formal military messaging systems [5]–[7]), where the sender does not know who is going to read the message in the receiving organization. When we need security in person-to-person service, we can easily use existing user-level security services such as PGP [8], [9] or PKI [10]–[12] tools. However, in the case of organization-to-organization service, such user-level security cannot be used. When sending, the sender does not know whose cryptographic key should be used to encrypt or sign the message, because the receiving individual is not yet determined by the sender. Figure 1 shows a typical example of organization-to-organization messaging services. A user (drafter) can initiate communication by writing or revising a message. Before the message is sent to another organization, the coordinator reviews the message and requires necessary changes in the message based on the organization's policy. If the message is ready to be sent, the releaser of the sending organization sends the message to the distributer in the receiving organization. Once the message is delivered, the message is distributed to the corresponding recipients based on the organization's current policy.

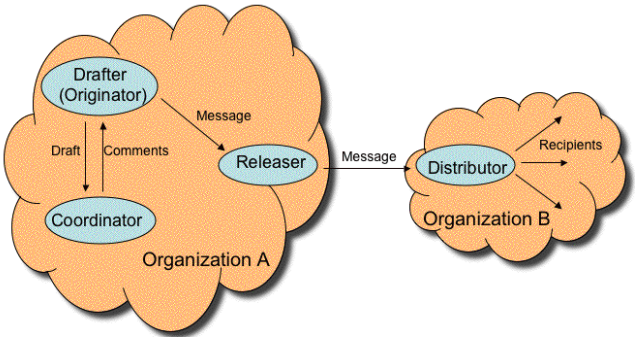

Fig. 1. An Example of Organization-to-Organization Messaging Services.

In this paper we focus on the message protection in the organization-to-organization service, while many other researchers have worked on the person-to-person service [13]–[18]. Our approaches can provide effective integrity verification, tracking mechanisms, and confiden-

tiality at the user-level in a scalable manner with fine granularity. Furthermore, we use a set of session keys for each message to provide more secure communications, maintaining scalable key management by employing a key hierarchy. We describe the mechanisms of our approaches and show their feasibility by describing a prototype system that we developed.

II. Related Work

A. Boundary Control

The technical solution for preventing unauthorized release of sensitive information between different domains is referred to as *boundary control*. A boundary controller is a software solution designed to enforce the rules and policies of an organization to control the information flow between the organization and the outside world, as well as between internal units. A high-accuracy boundary controller is essential in supporting the information sharing that is required within an organization where there are groups that need to share some, but not all, information with different levels of security. Technically, a boundary controller is an automated guard whose generic operation is depicted in Figure 2. For future tracking of transactions, it maintains log files for both sending and receiving organizations.

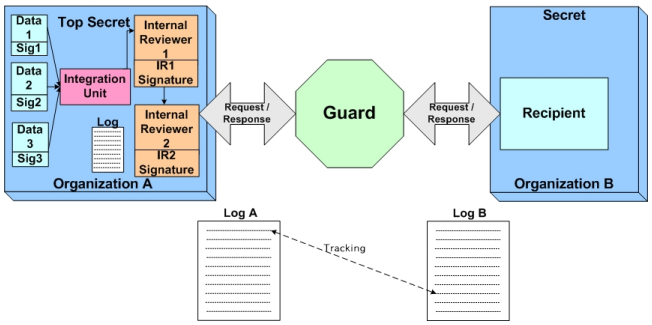

Fig. 2. A Generic Operation of Guard.

The Advanced Research Guard for Experimentation (ARGuE [19], [20]), developed by the NAI labs, is one of the working models of boundary controllers. Once a client initiates a request, it is sent to the ARGuE where the request is processed, and if it does not violate any of the policies, it is sent to the server on the other end. The processed result is sent back to ARGuE where verification of the security policies takes place and the response is given to the other end. Simultaneously, logs are developed and maintained for each and every transaction that takes place between the organizations.

The Air Force Rome Laboratory (AFRL) developed the Imaginary Support Server Environment (ISSE) Guard [21], [22]. ISSE provides a secure interface in the transaction of softcopy information between two hierarchically varying divisions within local or wide-area networks. The ISSE Guard is built in with Common Guard Interface (CGI). The ISSE Guard supports bi-directional traffic from High to Low or from Low to High security clearance. The ISSE Guard provides the functionality required for securely connecting, validating, downgrading, and transferring information between systems and networks operating at different security clearance levels. It permits secured transaction of digital images, files, and texts between organizations of different security clearance levels by downgrading the information, if transferred from organizations of high security levels to organizations with low security clearance. These are built in with email and file-scanning, sanitizing, and downgrading tools that work on XML-based products.

The Cross-Domain Solution (CDS [23]) was developed by the Department of Defense, and the Department of Navy. CDS was developed to send and receive data between different levels of organizations. It is a blend of technologies (i.e., guard, components), policies (i.e., rules, instructions, and procedures), and threat environments (i.e., reliability, trust level, and security clearance) in which it should be deployed. NetSec [24] developed a CDS system that offers high assurance for data transfer, but its data transfer takes place in one direction alone.

The Message Analysis Downgrade and Dissemination (MADD [21]), developed by AFRL, is a product developed to sanitize or downgrade the message and then pass it on to the user at the other end. This works only for well-formatted messages. It incorporates the message extraction technologies developed by AFRL to enable automated filtering, downgrading, sanitization, and dissemination, which are in the USMTF (US Message Text Format [25]).

B. Key Management

Alk and Taylor [26] proposed a cryptographic approach in a tree hierarchy based on modular exponentiation, which is the basis of current public-key cryptosystems [27]. This approach is effective for a key hierarchy that is an arbitrary partial ordering, but expensive regeneration of all the other existing values is required when a new class is added to the tree. Their modular exponentiation is not scalable, because it will require many calculation changes to be made in the rest of the system in order to accommodate this new depart-ment within the system. The modular exponentiation method cannot be adapted for the entire system, as there are possibilities of the same key value being generated since all the values are generated using one large number N. In addition, a large amount of storage space is wasted in storing the key values of the nodes.

Sandhu [28] proposed a key generation scheme for a tree hierarchy considering the access control problem in a system where users and information items are classified based on their security classes. In this scheme, different one-way hash functions are used to generate child keys from parent keys. One-way hash functions are easily computable, but it is computationally difficult or infeasibly difficult to invert them [29]. One-way hash functions are selected based on the hierarchical names of the child nodes. The main advantage of this scheme is that if a new

child node is added to the hierarchy tree, then the keys for its parent nodes need not be recomputed. Thus, changes to the hierarchy are conveniently accommodated by the scheme. This makes Sandhu's key generation scheme much more scalable than the Alk and Taylors [26] modular exponentiation method. Although Sandhu's scheme is faster than the modular exponentiation method, the chief disadvantage of his tree hierarchy approach is that the time it takes to generate the necessary keys for every node is excessive when the tree size is very deep (i.e., tall).

Zheng et al. [30] proposed an approach based on partial ordered sets, called posets. The authors identify two key generation approaches: one that provides an indirect access to the child nodes, and a second that provides direct access to the child nodes in the hierarchy tree. The indirect access technique requires a user to traverse all the intermediate tree nodes to calculate the key for a low-level node. The direct access approach allows direct key calculation of a low-level node without traversing other intermediate nodes.

Our key management approach is more related to Sandhu's approach, which is used in Zheng's direct-access method. We extend the one-way hash-function-based approach with the concept of session keys and their secure distribution for a trusted message system.

III. Operational Scenarios

As we mentioned in Section I, we focus on the message protection in the organization-to-organization service in this paper. When Alice is sending a message to Bob in a different organization, Alice composes the message and then digitally signs it. Optionally, the message can be encrypted for providing confidentiality. The signed packet is sent to the guard, where the contents of the message are verified with current policies that are set for message transfer. If the message was encrypted, the guard should be able to decrypt it by using the corresponding key. It is also possible for the guard to encrypt the message after verification. In most currently available approaches, if all the contents are allowed for transfer by the policies, then the guard strips out Alice's signature and appends its own signature. This message with the guard's signature is sent to Bob. Subsequently, Bob verifies the guard's signature and retrieves the contents. Typically, Bob does not need to verify that the message was originally signed by Alice, because Bob trusts the guard. If the message was encrypted, Bob can decrypt part of the message based on his security level. In the meantime, the guard creates entries in the log files for future reference purposes.

The operation can be extended with internal reviewers. An internal reviewer (IR) verifies the individual data (message) sent by each user and compiles them into a single data unit before sending it to the guard. Multiple internal reviewers can verify the data before sending it to the guard. Figure 2 illustrates how the data is integrated when we have three users compiling data to be sent to the other organization. The integration unit combines these messages and sends them to IR1. The integration unit strips out the signature of the preceding users and holds just the signature of the last user. This procedure is followed until it reaches the final user. After the message is integrated into one unit, it is sent to the IR1. When the data come out of IR1, the message will have only IR1's signature. Similarly, when the message comes out of IR2, the signature of IR1 will be stripped out and the signature of IR2 will be appended. Once both internal reviewers are done with their verification, the message with the IR2's signature is sent to the guard.

This conventional operation might be the simplest solution when a message is written by a single user and user-level verification is not required at the time of message receiving. However, when multiple users, especially with different security levels, are involved in the contents of the same message before it is transferred to the final receiving organization, it is challenging to provide and verify message integrity in a scalable manner with fine granularity. When message confidentiality is needed, the scalability problem becomes more critical in terms of key management. Furthermore, it is not always possible for the recipient to track the users who are involved in the message creation on the senders' side. Although the transactions are logged, it can still be difficult to track especially when the log file of the senders' organization is maintained by a different administration, which is not unusual. To overcome these problems, we propose the following advanced operational scenarios for providing message integrity and confidentiality.

Advanced Operation:

Each user who is involved in the message signs and/or encrypts his or her portion individually. After the guard verifies the compiled message and individual signatures, it attaches its signature to the message. Optionally, the guard can encrypt portions of the message by using different keys based on their security levels. The final message includes the guard's signature as well as the individual signatures. Later, the recipient can verify the individual signatures and decrypt some portions of the message based on his or her security level by using the corresponding keys.

This advanced operation will provide effective integrity verification and tracking mechanisms at the user level. Furthermore, it increases service granularity. However, it will also introduce the scalability issue in terms of signature maintenance and key management. In the following sections, we describe how we can support the above proposed scenario in a scalable manner with fine granularity .

IV. Scalable and Fine-Grained Digital Signature Schemes

When we have multiple users compiling a single message, including shared contents, metadata, policy, and so on, the integrity of the contents created by individual users needs to be maintained in a scalable manner

with fine granularity. There is an urgent need for new mechanisms in a trusted content-sharing environment to support multiple signers of the same message, which can be dynamically updated, with autonomous protection and maintenance mechanisms. In our previous work we identified and compared three different binding mechanisms of digital signatures, including monolithic, autonomous, and chained schemes [31], [32]. The original work was designed and implemented for digital certificates. In this paper we apply the digital signature schemes with extension to the organization-to-organization messaging services to support the advanced operation described above that provides fine-grained and scalable user-level integrity to messages.

A. Monolithic Signature Scheme

Of all the signature schemes, the simplest case is when the contents could be signed by one single authority. We define such case as *monolithic*, depicted in Figure 3. In this case, however, it is difficult to track the integrity at the user level, especially when multiple users with different security levels are involved in the same message. In other words, it is hard to check who provided specific contents to a message and to support a fine-grained integrity check unless a log file is maintained with detailed user-level histories. For tracking, the system has to go through the log files generated during the transactions and then trace the perpetrator. However, maintaining a powerful logging database itself, which should be protected in a sensitive system, is not a trivial work. Even if such a database exists, it usually takes a long time to track the necessary data, and the accuracy of the tracking depends on the logging mechanisms and information. Sometimes, the message-sending organization does not allow other organizations (including the receiving organization) to access its log files. Furthermore, in a traditional scheme, each time a new portion is added to the message, the signer should sign the entire message. This approach is neither efficient nor scalable for a large messaging system, especially if different portions of the same message need to be signed by different authorities. Finally, generating and maintaining log files that can support our advanced operation become complex, especially when messages are merged and separated frequently by users with different security levels.

B. Autonomous Signature Scheme

In the *autonomous* scheme, depicted in Figure 4, a binder of the original contents and the new message are digitally signed by an authorized person. A binder can be any unique portion in the original message. For instance, the message ID of the original message can be a binder to other linked messages in a general case. It is also possible to link a specific portion of the message to others. For instance, in Figure 4, if Contents X and Y in the new message are related to Contents A in the original message, Contents A would be chosen as the binder

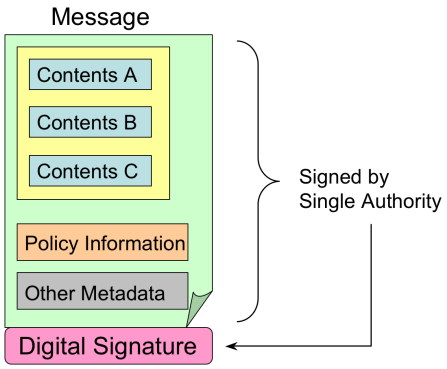

Fig. 3. Monolithic Signature Scheme.

between the messages to maintain their relationship. The main advantage of the autonomous scheme is that the binders and contents are loosely coupled. In other words, even after the original and new messages are linked by the binder, we can still modify the other portions of the original contents without breaking the link to the new message as long as the binder and the corresponding signature remain the same. One binder can be used to link different messages and the same message can be linked by different binders (a many-to-many relationship). The linking mechanism is analogous to the situation where a person can use any ID card to prove that he or she is the owner of his or her credit cards, as long as the names on the credit card and ID card match. This scheme provides fine granularity and scalability because only relative portions of a message are linked to new messages by digital signatures.

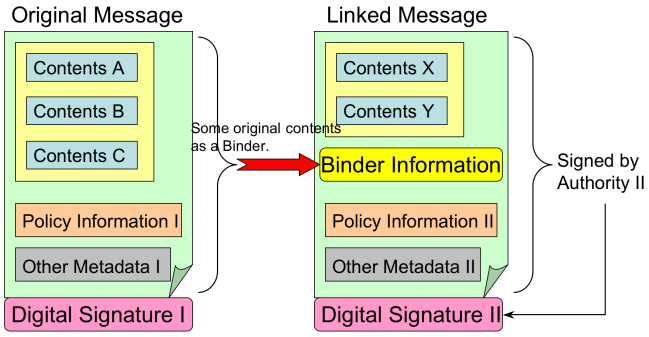

Fig. 4. Autonomous Signature Scheme.

Once the entire message is compiled by the autonomous scheme, the message is sent to the Internal Reviewer, who verifies each and every part of the message to confirm that the message does not violate company policies. After this screening process the integrated message is sent to the guard. If the message does not violate policy settings, then it is allowed to pass through to the other organization. When the user from the other organization requests this message, the guard annexes its own signature by the autonomous scheme just as the users did on the message. If the requesting organization is in a lower security clearance, then the guard downgrades the message to the security clearance level of the requesting

domain. However, if there is any portion of the data that is violating policy settings, it is immediately sanitized or downgraded. These kinds of activities are recorded in the log file. After the filtering and scanning process, the file is passed on to the requesting organization.

C. Chained-Signature Scheme

The *chained* scheme, depicted in Figure 5, is a particular case of an autonomous scheme, in which we use the digital signature of the original message as a binder. In this case the binders and other linked messages are tightly coupled, which means that once the original and new messages are linked by the binder (i.e., the signature of the original message), we cannot modify any portion of the original message without breaking the link to the previous signatures. Each signed entity can produce only one unique binder, digital signature, with the same key in the chained scheme. Once the digital signature has been generated, we cannot change any portion in the signed message without breaking the integrity of the entire original message, while one signature can be linked to many other new messages (a one-to-many relationship). The linking mechanism is analogous to the situation in which a person should present his or her particular ID (such as a passport) in a foreign country to prove that he or she is the owner of his or her credit card because that country does not understand or trust other IDs such as a driver's license card issued by another country.

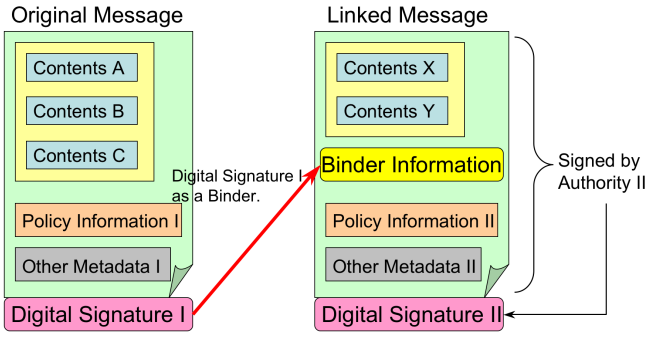

Fig. 5. Chained-Signature Scheme.

Since we use the signature of the original message as the binding information, the flexibility of adding new contents in the compiled message is going to be stricter than in the autonomous scheme. However, at the same time, it provides strong integrity verification and it still provides a finer granularity than that of the traditional approaches. Even a slight modification in the header or body part of the message will be reflected in the signature. If we make any modification in the original message, then the corresponding signature should be regenerated in the message that follows. In this scheme the messages are tightly coupled. When the Internal Reviewer verifies the message before sending it to the guard, it can easily determine the amendments made by other users. The guard annexes its own signature by the chained scheme just as the users did on the message. The rest of the functionalities of the guard are similar to those described in the autonomous scheme.

V. Scalable and Fine-Grained Key Management

A. Hierarchical Key Structure

As a sensitive organization, we consider a military organization to demonstrate our proposed ideas. In such an organization, typically, each user-group can be categorized according to its privileges (i.e., clearance). A group with the same privileges can be represented by the following set of variables: {U, L, P, S, K_i, R}, where U = User, L = Clearance Level, P = Partition, S = Session Number, K_i = Session Key for the clearance level i, and R = User-Key Relation.

In our case, a session number is a message ID. The variable R in the expression is an important part of the expression, as it concerns the users and the keys that they possess. This mathematical expression binding user and key is shown bellow.

$$\text{User } U \text{ owns a key } K_i \text{ if and only if } (U, K_i) \subset R$$

Since we are focusing on formal military message services (described in Section I) that exchange messages between organizations rather than between individuals, usually, the system does not determine a particular recipient when the message is being created. This implies that we should not use simple user-dependent security mechanisms. For instance, a sender should not encrypt a formal military message by using a particular receiver's key. Instead, the message should be encrypted by a group key of the sender side (e.g., a group key for a secret level) so that any recipient in the group (e.g., secret level or higher) is able to decrypt the message. Technically, we could use an independent key for each security level. However, this may increase complexity and cause a significant scalability problem, especially when we use different key sets for different messages. In our approach, therefore, we consider a simplified key hierarchy that falls into totally ordered sets: Top secret $\succeq$ Secret $\succeq$ Classified $\succeq$ Unclassified. This means a user with top-secret privileges will have the ability to generate keys for secret, classified, and unclassified. Similarly, users with secret privileges will have the ability to generate keys for classified and unclassified, but will not have the ability to generate a key for a top secret. Basically, we use a hash function to generate the keys. In this way we can generate keys for different security levels of users, while the same function is applied in all of the nodes that generate keys for the lower lying nodes. The nodes in the unclassified section (i.e., the lowest security level in our example) will not be able to generate any other keys. Therefore, it is unnecessary to distribute the function to the leaf nodes. When one adds a new node, one must determine that node's security-level to assign it the appropriate group key.

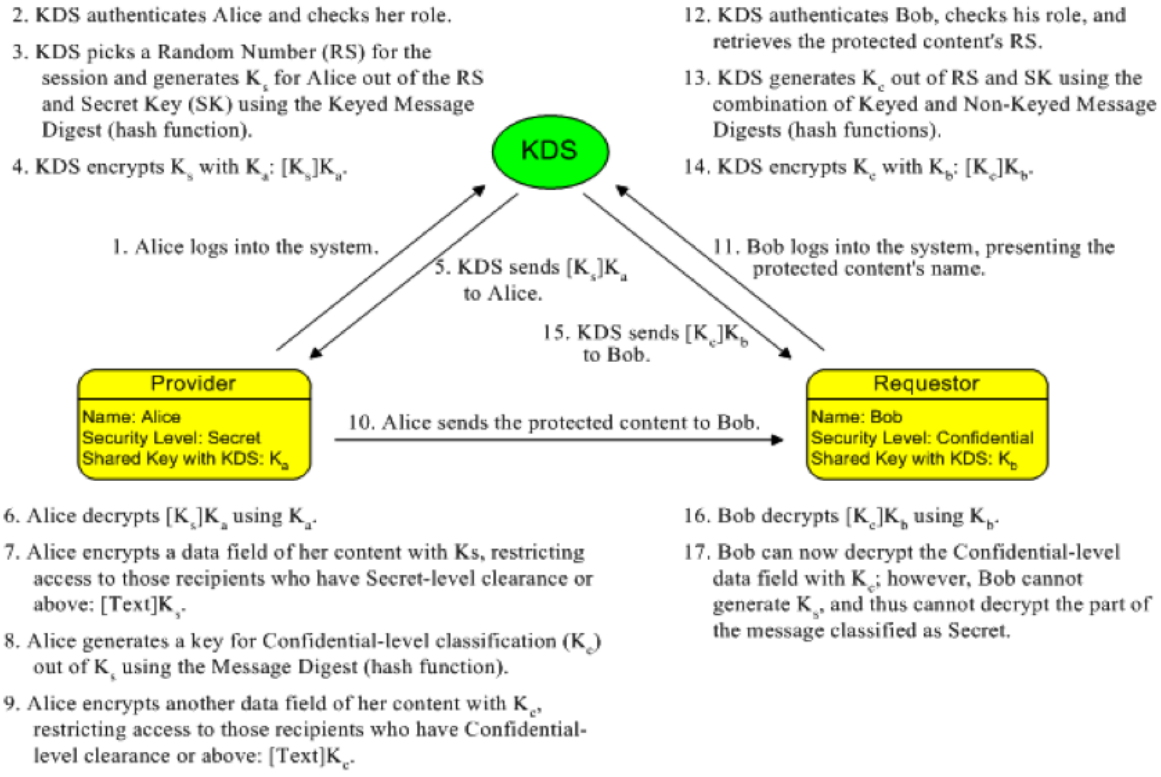

Fig. 6. Fine-Grained and Scalable Key Management Scheme.

B. Key Management Mechanisms

In our key management scheme, each security level is assigned a group key, and higher level keys can be used to obtain (generate) lower level keys. Technically, there is a centralized key distribution system (KDS) that initiates the session-key distribution services for each session (i.e., each message in our case) to individuals based on their clearance levels. It picks a secret random number, called secret session number (SSN), for each message and generates the highest session key (e.g., top secret) for the message by using the hash function. Alternatively, we can use a keyed message digest to generate the highest key for the message. In this case we can make the random number public, called public session number (PSN), while a master secret key is required for the KDS. Otherwise, anybody could generate child keys by using the publicly known hash function and the PSN. More details are described as follows.

Each message has a unique message ID, which is mapped to a random number RS (either SSN or PSN). This mapping information is recorded in the KDS. We do not need to store the session keys in the KDS because the session keys can be generated on-the-fly if necessary. Consequently, the user can generate the corresponding lower level keys. This saves storage cost and enables more secure and scalable key management than conventional approaches. The corresponding session keys based on the user's clearance are generated out of the highest session key, which is generated by the KDS and is securely distributed to the user. Lower level keys can be generated by the KDS or the user autonomously, based on the user's clearance. We assume that each user has a shared secret key with the KDS. Hence, the KDS encrypts the highest session key for the user with the shared secret key, and sends it to the user. The user can decrypt the encrypted session key, using her shared secret key with the KDS. The sessions are totally independent of each other. This means that the higher-level keys in one session cannot generate lower-level keys in different sessions, thus ensuring that the key issued to the user works only for the message that the user is trying to view.

Figure 6 illustrates detailed procedures for either using an SSN with a publicly known hash function or a PSN with a keyed message digest. For simplicity, we use the former case in the following description, where only the KDS picks up and knows the message's SSN, although we implemented the latter case, where the message's PSN is public and the KDS's master secret key is required to generate the highest-level session key out of the PSN using a keyed message digest function. In this example, there are two users that will communicate using our proposed ideas. Suppose the sender (Alice) has a secret clearance, and possesses a shared key, K_a, with the KDS. The receiver (Bob) has confidential clearance and possesses shared key, K_b, with the KDS. Initially, Alice logs into the KDS after authentication and requests a session key. Every time a session key is needed, the KDS starts with the session's (i.e., the message's) SSN that it previously acquired. When generating a session key, the KDS uses the random number (RS), SSN in this example, and other background information, such as

the name of the classification level, as input for the hash function. The process works only one way, meaning that lower keys can be generated from higher keys, but not the reverse. To generate the secret-level session key, K_S, for Alice, the KDS performs the following consequent operations.

A1. K_T = *H(SSN, Top-secret)*;
KDS generates a top-secret-level session key
A2. K_S = *H(K_T, Secret)*;
KDS generates a secret-level session key

Once the proper level key is generated, the KDS encrypts K_S with Alice's shared key, K_a, and sends the encrypted data to Alice. The KDS never sends any information about K_T because Alice's clearance is Secret. Since Alice knows K_a, she can decrypt the data to obtain the secret-level session key, K_S. Alice can then use K_S to encrypt the secret-level contents in the message she is writing. Alice also has the ability to create lower-level keys for lower-level encryption, which is important when she needs to work on contents that possess lower classification levels. For instance, she can obtain the confidential-level key by applying the hash function on K_S and other background information as follows:

A3. K_C = *H(SSN, Confidential)*;
Alice generates a confidential session key

Once K_C is generated, Alice can encrypt that part of the message with K_C.

After the message and its encrypted contents have been sent to the recipient's organization, those users with the required privilege are allowed to access the correspond-ing parts of the message. In our example, Bob logs into the KDS and requests his session key for the message, presenting the message ID. The KDS maintains the mapping information between the message ID and the corresponding SSN so that it finds the message's SSN and generates the confidential-level session key, K_C, for Bob, whose clearance is Confidential. To generate K_C, the KDS performs the following operations:

B1. K_T = *H(SSN, Top-secret)*;
KDS generates a top-secret-level session key
B2. K_S = *H(K_T, Secret)*;
KDS generates a secret-level session key
B3. K_C = *H(SSN, Confidential)*;
KDS generates a confidential session key

Once the key with the proper security level for Bob is generated, the KDS encrypts K_C with Bob's shared key, K_b, and sends the encrypted data to Bob. Since Bob knows K_b, he can decrypt the data to obtain the confidential-level session key, K_C. Bob can then use K_C to decrypt the confidential-level contents in the message. Bob can only view the contents of the message encrypted with K_C (not with K_S) because he cannot generate the K_S based on his clearance.

VI. Implementation

A. Overview

Based on the advanced operational scenario and the support mechanisms described in the previous sections, we developed a prototype. The prototype was implemented in the Java remote method invocation (RMI) platform with XML digital signatures [33]. The RMI framework provides a platform for developing distributed client-server applications. We employed the Java cryptographic extensions (JCE) APIs in our implementation. JCE is a set of packages that provides framework and implementations for encryption, key generation, key agreement, and message authentication code (MAC) algorithms. JCE integrated into the Java 2 SDK. The KDS (Key Distribution System) is implemented by KMS (Key Management Server) in our implementation that is an RMI server object. The users are mapped as corresponding RMI clients. The entire application was developed using the XML Security Suite in Java developed by the IBM Alpha Works. We also used the Java Security Package, which provides policy-based access control, X.509 v3 [34], [35] implementation of certificate interfaces, and tools for creating and managing security keys and certificates. Figure 7 is the activity diagram of our implementation that shows the operations carried out in each module and the information between the modules.

The KMS initially registers itself with the RMI registry, a naming service in the Java RMI framework, to make it available to others. The User is configured as an RMI client. The User class module can get a reference to the KMS server object by querying the RMI registry. Once it gets a reference to the KMS server object, then the user class module can access the methods of the KMS just as it can access any other method calls. The user interface class module has a graphical user interface (GUI) that is implemented using Java Swing packages. This class module basically gets all the user inputs and commands through the GUI and passes them to the user class module for further processing and sends the processed result back to be displayed in the GUI.

Only authenticated users are supplied with the keys for that session. If the user, say Alice, requests the composing of a new message, the screen opens up, depending on her role and security clearance. The author has the privilege of downgrading her security level to compose a message for the domains of lower security levels. After that, Alice encrypts, signs, and then sends the message to the XML guard where the message is deposited in the file repository. Now, supposing another user, Bob, wants to open this message and modify a few fields. All that he needs to do is log into the system, then post a request to the XML guard. Here, the request is verified and his security level (i.e., clearance) is crosschecked with the security level of the message. If the security level of the

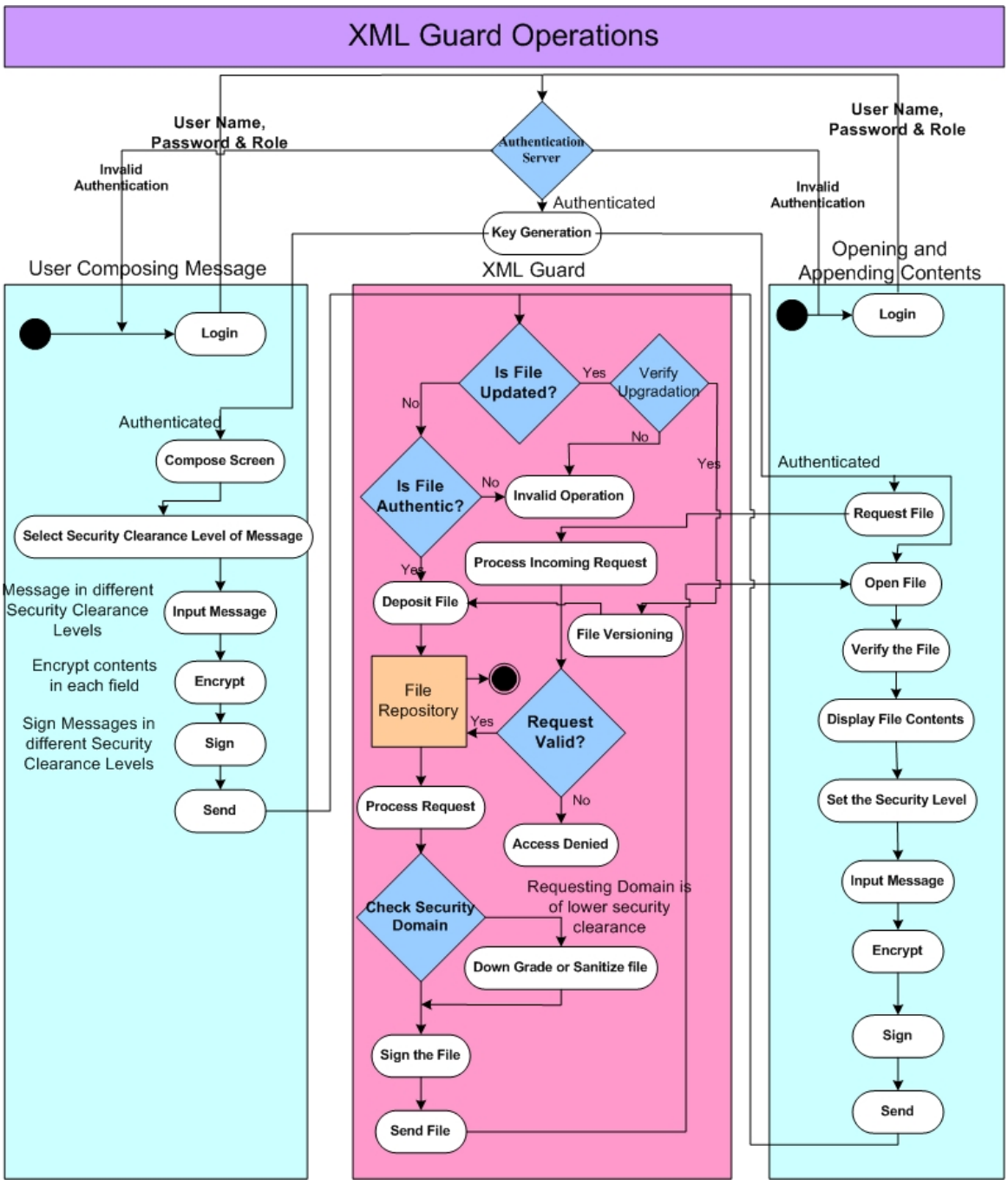

Fig. 7. Activity Diagram of XML Guard Operation.

message is higher than that of Bob, then the message is sanitized (i.e., the contents which are not to be revealed for the requesting domain are all stripped out) and then passed to the requesting user. If the security level of the message is lower than that of the requesting user, then no sanitizing operation is performed and the entire message is passed on to the requesting user. Similarly, if the security levels of the message and the requesting user are the same, then there is no sanitizing operation performed. Now, the modified file is encrypted and signed and then sent to the XML guard. In the XML guard, the modified file contents are verified and then deposited in the file repository. Before sending the message across the other domain, the XML guard signs the integrated message based on either the autonomous or chained schemes described in Section IV and passes it to the requesting user or organization.

B. Key Management Modules

According to our proposed ideas, the messages are generated for different levels of authentic users. In our implementation, all the messages are created and sent as Extensible Markup Language (XML [36]) files by the XML generator, and then transmitted to the receiving

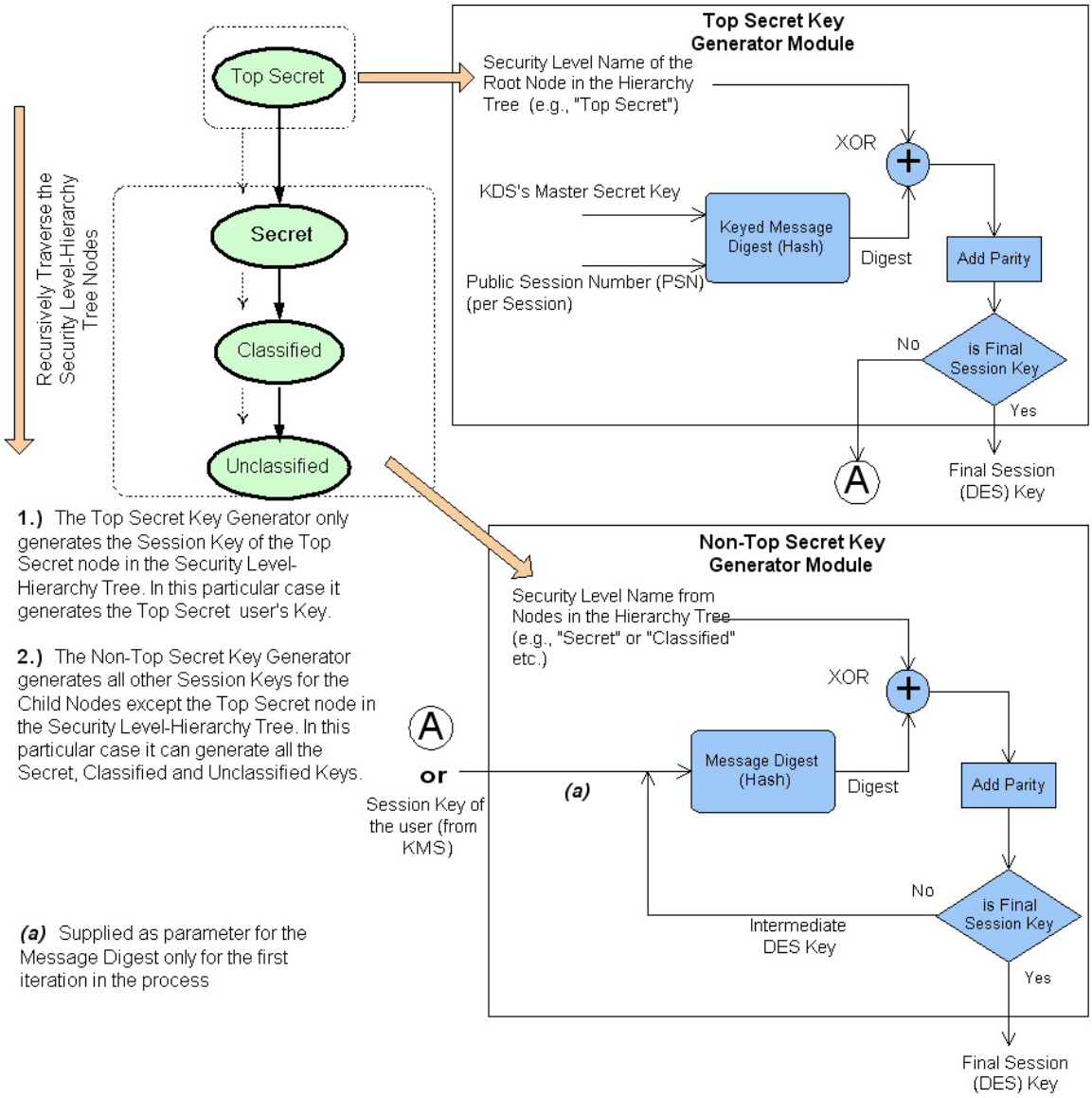

Fig. 8. Key Generator Modules.

organization. The Open Decrypted Message module opens the XML file that was sent, extracts the encrypted message, and then decrypts the message, if and only if the user has sufficient privilege to decrypt a message of that security level.

In our implementation we developed two key generator modules since we use a PSN and a keyed message digest algorithm for top-level key generation: one for the top-secret key generation and one for the rest of the keys. If we were to use an SSN and a non-cryptographic hash function, we would need only one key generator module. Figure 8 shows the inner details of our key generator modules and their functions.

The Top-Secret Key Generator module employs a message authentication code (MAC), also called a keyed message digest, which is basically a message digest with an associated key to generate the Top-Secret key of the root node (the highest Security Level in the tree). The basic principle is that someone with the same key can produce the same keyed message digest from the same input data. This module takes three parameters as input, and, subsequently, outputs the session key of the Top-Secret node. The keyed message digest takes a public session number (PSN) and the KDS's master secret key as input, which it then digests to generate a message digest. The key generator module process XOR with the digest and the security level name of the root node (e.g., Top Secret in our example), then generates a bit array as an output. The resulting bit array is then passed to a parity checker, where the proper parity is added to the key. We used a keyed message digest instead of a simple message digest in order to secure the process of digesting the RS. Even if a PSN is compromised or made known, an attacker needs the Secret MAC key to gener-ate the digest. This secret key can be stored in a secure location by adding more security to the whole process.

The Non-Top Secret Key Generator module employs

just a regular message digest. The basic principle behind this is that anyone with the same input text can supply it to the message digest and get the same output. As depicted in Figure 8, this module contains the same components as the Top-Secret Key Generator module does, except that message digest replaces the keyed message digest. The message digest takes just one parameter as input; in this case, it takes a key as an input and digests it into a bit array. The output bits are then XORed with the security level name of the current node in the hierarchy. Finally, the parity bits are set properly to produce a well-formed key. This process is applied repeatedly as we traverse the tree nodes recursively until we arrive at the destination node for which we need to generate the session key.

Both the KDS and user modules have an instance of the Non-Top Secret Key Generator module, but only the KDS has the additional instance of the Top-Secret Key Generator module. Again, this is because a user should not, under any circumstances, be able to generate the session key for the highest security level (Top Secret) in the hierarchy tree.

In our current implementation, both the KDS and user modules contain an instance of the Key Generator module. The only difference is that the user module does not have the root key generator function, because in no circumstances should the user be able to generate by itself the root session key—the highest level in the hierarchy. The user module has the component represented by only the lower-right box, while the KDS module has the component represented by both the lower-right and upper-right boxes. An alternative design or architectural change would be to implement the Key Generator modules and KDS in different locations: either distributed or centrally located. In our implementation, we use the distributed architecture. The key generation modules are distributed to the KDS and each user's machine. This architecture enables autonomous key generation in each user's machine, avoiding single-point-of-failure problems while provid-ing higher performance. However, the implementation of this distributed architecture is more complex, and key revocation is a challenging problem. The centralized architecture—where a central location (such as a KDS) has common key generation modules for the participants—is easier to implement and more effective for key revoca-tion than the distributed architecture, because all the keys are generated and managed from a central point. However, such a centralized architecture does not allow autonomous services, cannot avoid the single-point-of-failure problem, and decreases the quality of the performance. More details about this implementation are available in [37].

C. Digital Signature Modules

Table I summarizes the tradeoffs of the digital signature schemes based on the results of our experiments. We will begin with the level of convenience. In the monolithic scheme only one user compiles the entire message and hence the message integration is very easy. In the autonomous and chained schemes, however, we have multiple users integrating the message and they need to specify the binder information for the integration. In the chained scheme we use just the signature of the previous message, whereas in the autonomous scheme we could use any part of the header or message as binder information. The user determines which binder he or she is going to employ in the autonomous scheme. Therefore, relatively, it is more convenient for the user to integrate the messages in the chained scheme than in the autonomous scheme.

In terms of message discovery, which implies the ability to spot a particular message written by a specific author, it is fastest to retrieve the entire message in the monolithic scheme because there is no links between the messages. However, in the autonomous scheme, the messages are linked based on a shallow model with many branches. In the chained scheme, typically, the messages are linked through long chains. Therefore, it is slower to retrieve the entire message in the chained scheme than in the autonomous scheme.

As we described in Section IV, the binding strengths of the monolithic and chained schemes are tightly coupled, which means that even if one portion of the message is modified, the integrity of the whole message following it is lost as well. However, the binding strength of the autonomous scheme is loosely coupled, which means the contents in the messages can be modified without breaking the message links as long as the binder remains the same.

Branching of messages is not available in the monolithic scheme, very high in the autonomous scheme, and moderate in the chained scheme. In the monolithic scheme, a single user inputs all the contents in the same message, so there is no branching between messages. In the autonomous scheme, branching with other messages can be widely done via various parts of the messages as binder information. In the chained scheme, we can use only the signature of the previous message as binder information, so the messages are not widely branched, but they can be branched in a long chain.

In terms of control granularity, the autonomous scheme provides the finest granularity because part of the same message can be used as a binder that is linked to other messages. On the contrary, the monolithic scheme provides a course granularity because the entire message is signed by a single user and used as the minimum control unit. The chained scheme provides a finer granularity than the monolithic scheme because, even though we do not have much control on the original message, we can add new contents with fine granularity. Therefore, the level of scalability maps to the same order as the granularity of the signature schemes.

Finally, we consider the mapping relationships between linked messages. Obviously, there is only a self-mapping available within the same message in the monolithic scheme. In the autonomous scheme, various parts of the original message can be used as a binder, which can be

Characteristics	Monolithic	Autonomous Scheme	Chained Scheme
Level of Convenience	High	Medium	Low
Message Discovery	Medium	Low	Slow
Binding Strength	Tightly Coupled	Loosely Coupled	Tightly Coupled
Branching of Messages	N/A	Wide	Deep
Control Granularity	Coarse	Fine	Medium
Scalability	Low	High	Medium
Mapping Relationship	Self	Many-to-Many	One-to-Many

TABLE I
CHARACTERISTICS OF THE DIGITAL SIGNATURE SCHEMES.

linked to multiple other messages. Therefore, we define the mapping relationship as many-to-many in the scheme. In the chained scheme, only the signature of the original message can be used as a binder, but it can be linked to multiple other messages. Therefore, we define that mapping relationship as one-to-many.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we proposed novel approaches for the message protection in the organization-to-organization messaging service. Our approaches can provide effective integrity verification, tracking mechanisms, and confidentiality at the user-level in a scalable manner with fine granularity. Furthermore, we used a set of session keys for each message to provide more secure communications, maintaining scalable key management by employing a key hierarchy. We described the mechanisms of our approaches and show their feasibility by describing a prototype system that we developed. For our current prototype we used XML technology with Java security packages. For our future work, we plan to extend the system's capability for interoperability between different message formats and semantics via the application of ontology [38]–[40].

REFERENCES

[1] D. Bell and L. Lapadula, "Secure computer systems: Mathematical foundations," The MITRE Corporation, Bedford, MA," Technical Report, March 1973, mTR-2547.

[2] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[3] C. E. Landwehr, "Formal models for computer security," *ACM Comput. Survey*, vol. 13, no. 3, pp. 247–278, 1981.

[4] L. J. LaPadula and D. E. Bell, "Mitre technical report 2547, volume ii," *Journal of Computer Security*, vol. 4, no. 2-3, pp. 239–263, 1996.

[5] C. Heitmeyer, C. Landwehr, and M. Cornwell, "The use of quick prototypes in the secure military message systems project," in *Proceedings of the Workshop on Rapid Prototyping*. New York, NY, USA: ACM Press, 1982, pp. 85–87.

[6] C. E. Landwehr, C. L. Heitmeyer, and J. McLean, "A security model for military message systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 3, pp. 198–222, 1984.

[7] R. W. Shirey, "The defense message system," *ACM SIGCOMM Computer Communication Review*, vol. 20, no. 5, pp. 48–55, 1990.

[8] S. Čapkun, L. Buttyán, and J.-P. Hubaux, "Small worlds in security systems: an analysis of the pgp certificate graph," in *Proceedings of the New Security Paradigms Workshop (NSPW)*. New York, NY, USA: ACM Press, 2002, pp. 28–35.

[9] P. R. Zimmermann, *The Official PGP User's Guide*. MIT Press, 1995.

[10] K. P. Bosworth and N. Tedeschi, "Public key infrastructures - the next generation," *BT Technology Journal*, vol. 19, no. 3, pp. 44–59, 2001.

[11] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu, "Internet x.509 public key infrastructure certificate policy and certification practices framework," Network Working Group, , United States, Tech. Rep., 2003, rFC3647.

[12] M. R. Thompson, A. Essiari, and S. Mudumbai, "Certificate-based authorization policy in a pki environment," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 4, pp. 566–588, 2003.

[13] I. Brown and C. R. Snow, "A proxy approach to e-mail security," *Software—Practice & Experience*, vol. 29, no. 12, pp. 1049–1060, 1999.

[14] M. Ferris, "New email security infrastructure," in *Proceedings of the New Security Paradigms Workshop (NSPW)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 20–27.

[15] I. Fette, N. Sadeh, and A. Tomasic, "Learning to detect phishing emails," in *Proceedings of the 16th international Conference on World Wide Web*. New York, NY, USA: ACM Press, 2007, pp. 649–656.

[16] S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller, "How to make secure email easier to use," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM Press, 2005, pp. 701–710.

[17] S. T. Kent, "Internet privacy enhanced mail," *Communications of the ACM*, vol. 36, no. 8, pp. 48–60, 1993.

[18] A. Levi and Çetin Kaya Koç, "Inside risks: Risks in email security," *Communications of the ACM*, vol. 44, no. 8, p. 112, 2001.

[19] J. Epstein, "Architecture and concepts of the argue guard," in *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC)*. Washington, DC, USA: IEEE Computer Society, 1999, p. 45.

[20] E. Monteith, "Genoa tie, advanced boundary controller experiment," in *the 17th Annual Computer Security Applications Conference (ACSAC)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 74.

[21] *ISSE Guard*, the Air Force, October 2004, http://www.if.afrl.af.mil/tech/programs/isse/.

[22] *Information Support Server Environment (ISSE) Guard*, GlobalSecurity.org, 2006, http://www.globalsecurity.org/intell/systems/isse-guard.htm.

[23] *Cross-domain solutions*, Galois, 2006, http://www.galois.com/xdomain.php.

[24] *NetSec: Managed Security. Business Relevance*, NetSec, 2007, http://www.netsec.net/content/index.jsp.

[25] *US Message Text Format, Electronic Documentation System*, DISA/JIEO, 1998, mIL-STD-6040.

[26] S. G. Akl and P. D. Taylor, "Cryptographic solution to a problem of access control in a hierarchy," *ACM Transactions on Computer Systems (TOCS)*, vol. 1, no. 3, pp. 239–248, 1983.

[27] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and publickey cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[28] R. S. Sandhu, "Cryptographic implementation of a tree hierarchy for access control," *Information Processing Letters*, vol. 27, no. 2, pp. 95–98, 1988.

[29] J. Rompel, "One-way functions are necessary and sufficient for secure signatures," in *Proceedings of the Twenty-Second Annual*

ACM Symposium on Theory of Computing. New York, NY, USA: ACM Press, 1990, pp. 387–394.
[30] Y. Zheng, T. Hardjono, and J. Seberry, "New solutions to the problem of access control in a hierarchy," Department of Computer Science, University of Wollongong, Tech. Rep., 1993, technical Report Preprint 93-2.
[31] J. S. Park, "Towards secure collaboration on the semantic Web," *ACM Computers and Society*, vol. 32, no. 6, June 2003.
[32] J. S. Park and R. Sandhu, "Binding identities and attributes using digitally signed certificates," in *Proceedings of the 16th Annual Conference on Computer Security Application (ACSAC)*, New Orleans, Louisiana, December 11-15, 2000.
[33] *XML-Signature Syntax and Processing*, W3C, February 2002, http://www.w3.org/TR/xmldsig-core/.
[34] *ITU-T Recommendation X.509*, March 2000, iSO/IEC 9594-8:2001.
[35] M. Myers, C. Adams, D. Solo, and D. Kemp, "Internet X.509 certificate request message format," Network Working Group, United States, Tech. Rep., 1999, rFC2511.
[36] *Extensible Markup Language (XML)*, W3C, 2007, http://www.w3.org/XML/.
[37] J. S. Park and G. Devarajan, "Fine-grained and scalable approaches for message integrity," in *Proceedings of the 40th Hawaii International Conference on Systems Sciences (HICSS-40)*, Big Island, HI, January 3-6, 2007.
[38] T. Edgington, B. Choi, K. Henson, T. Raghu, and A. Vinze, "Adopting ontology to facilitate knowledge sharing," *Communications of the ACM*, vol. 47, no. 11, pp. 85–90, 2004.
[39] S. Kaushik, D. Wijesekera, and P. Ammann, "Policy-based dissemination of partial Web-ontologies," in *Proceedings of the Workshop on Secure Web Services*. New York, NY, USA: ACM Press, 2005, pp. 43–52.
[40] V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg, "Ontology in information security: a useful theoretical foundation and methodological tool," in *Proceedings of the New Security Paradigms Workshop (NSPW)*. New York, NY, USA: ACM Press, 2001, pp. 53–59.

Joon S. Park is an assistant professor and the director of the Laboratory for Applied Information Security Technology (LAIST) at the School of Information Studies (iSchool) at Syracuse University, Syracuse, New York. Before he joined the iSchool in 2002, he did research in information security at the U.S. Naval Research Laboratory (NRL)'s Center for High Assurance Computer Systems (CHACS). He completed his doctorate at George Mason University, Fairfax, Virginia, in 1999.

Ganesh Devarajan is a security researcher at TippingPoint's Digital Vaccine group. Currently he focuses on SCADA security research and other application-based security services. He was a research member at the Laboratory for Applied Information Security Technology (LAIST) at Syracuse University in New York. He was involved in research activities for XML security and digital signature schemes. He holds a masters degree in computer science from Syracuse University.