A Policy-Driven, Content-Based Security Protocol for Protecting Audit Logs on Wireless Devices

Wassim Itani, Ayman Kayssi, Ali Chehab, and Camille Gaspard
(Corresponding author: Ali Chehab)

Department of Electrical and Computer Engineering, American University of Beirut
Beirut 1107 2020, Lebanon. (Email:chehab@aub.edu.lb)

(Received Sept. 3, 2005; revised and accepted Oct. 11, 2005)

Abstract

This paper presents PATRIOT, an optimized, policy-driven security architecture for protecting the confidentiality and integrity of audit log files on wireless devices. PATRIOT is based on a set of well-known cryptographic protocols and is designed to suit the limited nature of wireless devices. It offers a policy-driven, customizable security model and specifies a flexible, multi-level, and fine-grained encryption methodology that provides the suitable security strength without compromising performance. PATRIOT is designed in a platform-neutral manner and it can be deployed on a wide range of wireless devices and operating systems.

Keywords: Audit logs, customizable security, policy-driven security, security

1 Introduction

On computer systems, audit log files contain sensitive information whose privacy and integrity must be protected against any malicious reading, modification, or deletion. Audit logs contain valuable information such as event dates and times, event IDs, event descriptions, transactions, business records, and other confidential data. This makes audit log files real "honey pot" resources for attackers trying to steal confidential information, modify it, delete it, or even add false and bogus transactions and actions to the audit logs for gaining business or personal advantage.

The audit log security system described in this paper is not a system that prevents an attack from occurring, but rather a system that guarantees the confidentiality of the log data generated prior to the attack, and that detects any malicious modification, deletion, or insertion to this data.

Many technologies and systems exist for protecting the confidentiality and integrity of audit logs; however, none of these systems take into consideration the issues and particularities of protecting audit logs on small and limited-resource wireless devices. These devices are usually characterized by their small size, which makes them very easily lost, stolen, and compromised. This fact raises real concerns about the privacy of the sensitive information stored on the device and in particular that generated and stored in log files. Moreover, wireless devices vary greatly in capabilities and resources. For this reason, the protocols used in securing audit logs on these devices have to be designed specifically for operation in wireless environments and must address the needs and requirements of a large variety of devices which are, in majority, severely constrained in terms of processor speed, memory resources, network bandwidth, battery, and storage capacity. This diversity makes the implementation of a single security standard to encompass the whole device range infeasible. A least-common denominator security standard that targets devices with limited memory and slow processors would be unfair for powerful devices and would not meet their security requirements, and in the same sense, a security standard that addresses high-end devices would neither fit nor perform efficiently on limited-resource devices. What is needed, therefore, is a security protocol that can be customized and configured to perform the security operations flexibly; taking into consideration the memory capabilities and the processing power of the device and the specific security requirements of the application.

In addition, many companies today are providing their employees with Personal Digital Assistants (PDAs) and smart phones to access corporate data and networks and to perform business- and mission-critical operations. In this situation it is very important to have a security standard that guarantees to the company authorities that the actions performed by their employees are legitimate and legal, and that the action description entries in the audit logs of the devices are not tampered-with by the employee

or by others. Moreover, the same security service must allow the employee to prove to the company authorities that he or she has really carried out a certain business transaction by tracking the transaction entry in the audit log file and verifying the integrity of the audit log data.

This paper presents PATRIOT, a security protocol for ensuring the privacy and integrity of audit logs on wireless devices. PATRIOT doesn't prevent an attacker from modifying or deleting the audit log entries, especially when the attacker is the owner of the device; however, it prevents the attacker from reading log entries created before the time of the attack and it guarantees the detection of any malicious modification, deletion, or insertion made on the log entries generated prior to the attack. PATRIOT is based on well-known cryptographic protocols for securing audit logs [1, 11]; it simplifies some of their features and introduces the concept of content-based, multi-level encryption that secures data based on content and sensitivity rather that encrypting "everything". PATRIOT provides its confidentiality and integrity security services based on a configurable security policy that specifies several security-related attributes, classifies the fields of the log file based on sensitivity and content, and identifies the scope and strength of the encryption and hashing operations. All this contributes to decreasing the number of encryption and hashing operations and controlling their level which results in great flexibility and an overall performance improvement.

The rest of the paper is organized as follows. In Section 2, we describe PATRIOT's threat model. In Section 3, we give a brief review of related work dealing with securing audit logs. Then we move on to discuss the design and architecture of PATRIOT in Section 4. This will include an overview of the different components of the architecture. Section 5 describes the verification process of the log file on the wireless device. In Section 6, we present a formal and platform-independent mathematical analysis of PATRIOT's performance and show the cost savings offered over traditional log security systems [11, 12]. Section 7 provides an overview of a complete simulated implementation of PATRIOT on the Windows CE platform using the .NET Compact Framework [6]. The implementation section includes some platform-dependent performance results on the HP iPAQ Pocket PC such as the policy-parsing time and the encryption, decryption, and hashing rates. Section 7 also presents a performance comparison with a traditional security system that secures the log file by encrypting all its contents. Some conclusions are provided in Section 8.

2 Threat Model

The threat model we assume in this work is described as follows: we have a wireless device that is capable of performing wireless network interactions with some trusted server; initially, we assume that this device is trusted and that its software components and logging mechanisms are correctly configured and installed. At time t, this device is totally compromised by an attacker who tries to read, modify, or even delete the audit logs for gaining a personal or business advantage. The attacker could be the owner of the device (consider the scenario presented in Section 1 where a company offers its employees PDAs to perform mission-critical operations), or any hacker controlling the device locally, or remotely over the wireless network. The role of PATRIOT is not to prevent this attack from occurring, but rather to accurately identify it, to prevent any violation to the privacy of the log data generated before the time of the attack, and to detect any modification or deletion on the log entries created before time t.

3 Previous Work on Securing Audit Logs

The classical approach for protecting the integrity of audit logs is to write the logging data to write-once optical drives known as Write Once Read Multiple (WORM) drives, or to send it to a continuous-feed printer. This approach, in addition to being infeasible on a small wireless device, assumes that the WORM drive or the continuous-feed printer is not compromised by the attacker. Add to this the fact that continuous-feed printers are not suitable for printing high-volume logging and that it is nearly impossible to electronically analyze the output of these printers.

Another approach to protect audit logs from malicious tampering is to continuously send the logging data to one remote host (remote logging) or several remote hosts (log replication). This approach imposes a large load on limited-resource wireless devices and produces enormous network traffic and overhead that is not necessary in a wireless environment with a limited network bandwidth.

Bellare and Yee [1] introduced the "forward integrity" property which guarantees the integrity of the log data prior to the attack. Schneier and Kelsey [11] took a similar approach and presented a secure audit log system that relies on the presence of a trusted machine T. T's role is to provide the cryptographic key A, upon which the security of the whole system is based, and to verify the integrity of the log data. The paper also introduced the concept of log-level permissions which specifies the log entries that can be accessed by partially trusted users.

PATRIOT builds on these two protocols [1, 11] to provide a secure audit log system that suits the limited capabilities and resources of a wireless device. It simplifies some of their features (such as permission masks and verification by semi-trusted parties) and specifies a policy-driven architecture that supports a configurable, content-based, and fine-grained encryption and hashing scheme.

PATRIOT allows the verification of the audit log on the wireless device itself. This verification can be carried out by a validation auditor whose job is to install a verification software component on the device for validating the integrity of the log files. We assume that an attack

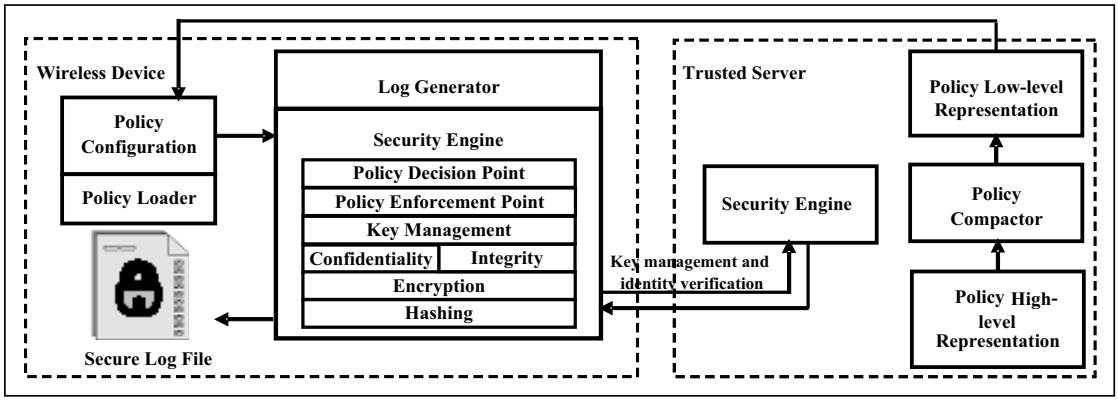

Figure 1: PATRIOT design and architecture

that tweaks the device's operating system to change the behavior of newly installed software components on the device is expensive for the attacker to contemplate.

4 PATRIOT Design and Architecture

This section provides an overview of the design and architecture of PATRIOT on the wireless device and on the trusted remote server. An abstract view of the major components of the security model is shown in Figure 1.

4.1 The Log Generator

The Log Generator component is the process responsible of generating log records on the wireless device. Each generated log record consists of a set of well-defined fields specifying precisely the action that occurred on the wireless device. It should be noted that PATRIOT doesn't depend in its operation on the number of fields or their respective data size. That is, the log record can consist of any number of fields each of which can have a variable data length.

4.2 The Security Policy

The security policy in PATRIOT specifies the security behavior and operation of the logging system. Its source information is present in a secure repository on the trusted server side and its high-level representation is configured by the authority responsible of the wireless device and the operations performed on it, such as a company authority monitoring and validating the actions performed by its employees' PDAs and smart phones.

PATRIOT's security policy consists of two main parts. The first part specifies a set of security-related attributes and configuration parameters, while the second part identifies the scope and strength of the encryption and hashing operations to be applied on the fields of each log record. The security-related attributes and configuration parameters specified in the first part are required by the Security Engine component on the wireless device to control the confidentiality, integrity, and key management operations. The Security Engine is the component responsible for taking security decisions and carrying out security-related operations to secure the output of the Log Generator. The following is a list of attributes supported by PATRIOT's security policy.

1) *Encryption_Algorithm*: specifies the symmetric encryption algorithm to be used for securing the privacy of the log records.

2) *Hashing_Algorithm*: specifies the hashing algorithm to be used in securing the integrity of the log file.

3) *Trusted_Server_URL*: specifies the Uniform Resource Locator (URL) of the trusted remote server.

4) *Key_Management_Algorithm*: specifies the key management algorithm for sharing the initial secret key between the wireless device and the trusted server.

5) *Secure_Log_URL*: specifies the location of the secure log file on the wireless device.

6) *Field_Count*: specifies the number of fields composing the log record.

7) *Field_Separator*: specifies the specific character(s) separating the different fields of a log record (usually this is a tabulation (TAB) character).

It is worth emphasizing that PATRIOT, in theory, can generically support any number of encryption, hashing, and key management algorithms. However, increasing the number of algorithms is practically infeasible due to the limited storage capabilities on current wireless devices. The second part in the security policy controls the level and scope of the security operations to be applied on the fields of the log records. According to this specification, the security of every field belongs to one of three security modes:

1) *A Secure_All mode*: this mode states that all the contents of the field must be secured, and specifies the strength of the encryption to be applied on this field. Four encryption levels are supported by PATRIOT: a High Security level which is equivalent to 256-bit AES [12] encryption (By AES encryption we mean an encryption complying with the AES standard as provided by the National Institute of Standards and Technology (NIST) and we don't specify the Rijndael algorithm itself); a Medium Security level which is equivalent to 192-bit AES encryption; a Low Security level which is equivalent to-128 bit AES encryption, and a No Security level.

2) *A Secure_Range mode*: this mode specifies the field sections to be secured in byte ranges together with the level of security to be applied on each range. The term "byte" in this context doesn't represent a physical unit of data storage but rather a logical unit whose specification depends on the representation of the field data being secured. This logical and generic representation is very essential since it allows the specification of these ranges without any dependency on the data encoding mechanism and representation.

3) *A Secure_None mode*: the field must be stored as is without any encryption.

Not all the log records generated by the log generator will be treated in the same way by the Security Engine. PATRIOT's security policy introduces the concept of log classes. According to this concept, every record generated will be secured depending on the log class it belongs to. The log class membership criterion is based on one of the fields of the generated log record satisfying a particular *regular expression*. If the generated log record doesn't satisfy any of the log classes specified in the security policy, it will be secured based on a default log class. Moreover, every log class specifies whether or not the fields of the log record satisfying this class are to be added to the hash chain that enforces the integrity of the whole log file. This is illustrated in Figure 2 which presents the structure of a typical high-level policy configuration. It should be noted here that PATRIOT leaves the issue of resolving any conflict, which may arise if a log record satisfies two different log classes, to the implementer. That is, the implementer may assign the log record to the first log class it satisfies in the policy configuration, to the default log class, or in any other implementation-specific way.

```
Encryption_Algorithm: Rijndael
Hashing_Algorithm: SHA-1
Trusted_Server_URL: fea.aub.edu.lb
Key_Management_Algorithm:DH
Secure_Log_URL: c:/system/securelog/securelog.txt
Field_Count:3
Field_Separator: TAB
//End of part 1, start of part 2
Log_Class_1 (Field1==regular expression 1)
        {
        Field1
            {
            Encryption: Secure_None  //Secure_None mode
                    Integrity_Enforcement:No
                     //field1 of log records belonging to this class
                     // will not be included in the hash chain
            }
        Field2
            {
    Encryption: <1-15 , High_Security>
                    <25 - 43, Low_Security >
                    //Securing byte
                    //ranges
        Integrity_Encforcement: Yes
            }
        Field3
            {
            Encryption: <1-4, No_Security>
                            <5-*, Medium_Security>
            Integrity_Enforcement:Yes
            }
    } // end of Log_Class_1
Log_Class_2 (Field3==regular expression 2)
        {
        Field1
```

Figure 2: Sample high-level policy configuration

4.2.1 Policy Compaction

The policy information is initially stored in a high-level format on the trusted server side. This high-level policy representation allows the policy authority to easily specify and customize the policy configuration using intuitive and flexible concepts and terms. Before delivering the policy configuration to the wireless device, there has to

be a mechanism that converts the high-level policy representation into a low-level binary representation that is understood by the wireless device's operating system. For this reason, PATRIOT introduces the Policy Compactor component. This component is responsible for parsing the high-level policy configuration file and converting it into a compact binary representation before transmitting it to the device. The policy compaction and optimization mechanism plays a major role in reducing network traffic (if the policy configuration is to be transmitted to the wireless device using the wireless network) and the storage and processing requirements on the wireless device.

4.2.2 Policy Loading

PATRIOT introduces the Policy Loader component for loading the low-level policy configuration on the wireless device. The Policy Loading mechanism and the process of getting the policy configuration from the trusted server to the wireless device may take different forms depending on some security considerations and assumptions that are discussed in the next section.

4.2.3 Policy Security

The security policy in PATRIOT contains sensitive information that controls the various security operations of the logging system. Any malicious modification to this information may lead to dangerous security attacks (consider the scenario where all the security modes are maliciously modified to the *Secure_None* mode or where all the security levels and the integrity enforcement parameters are nullified). Thus, assuring the integrity of the policy information, before and after loading it on the wireless device, must be given exceptional attention. Securing the policy configuration integrity before it is loaded on the wireless device can be easily accomplished by signing this policy information with the private key of the policy authority and appending the resulting digital signature to the policy configuration. This mechanism doesn't depend on the transport medium used. That is the policy configuration could be loaded to the device over the wireless network, using a physical storage facility such as a memory stick or a smart card, or it could be initially built into the wireless device by the device's manufacturer (after some coordination with the policy authority). When the policy configuration is loaded for the first time on the device, the Security Engine component checks its validity by verifying the digital signature appended to it using the policy authority public key.

Securing the integrity of the policy configuration on the wireless device, particularly in the memory of the wireless device, from malicious modification attacks is considered more challenging. This is due to the fact that the initiator of the attack may be the user of the device who has all the administrative privileges on this device. Let's consider the following alternatives: if the binary policy configuration is loaded from the trusted server over the wireless network, it can be appended to the Security Engine component as an external binary plug-in (after checking its integrity). However, in this scenario we must assume that the attacker is not capable of modifying this binary plug-in representing the policy configuration. To enforce this, memory locking mechanisms may be used to protect the policy configuration address space; however this depends on whether the device's operating system provides this capability through some Application Programming Interfaces (APIs). To relax the above assumption, the policy configuration could be sent to the device on a read-only memory (ROM) stick or on a tamper-resistant smart card, or it can be even burnt into the device's ROM initially by the device's manufacturer. In this case the Security Engine component must be configured to only load the necessary policy configuration portions from the ROM to the RAM whenever they are required, and to purge these portions from the RAM once it is done with them. This reduces the chance that the policy configuration is maliciously written when it is in the RAM of the device. A third alternative is to use a tamper-resistant cryptographic coprocessor to run the Security Engine and to store the policy configuration. The concept of validating software outputs using a cryptographic coprocessor is extensively described in [10].

4.3 The Security Engine

The Security Engine is the component responsible of securing the integrity and confidentiality of the log records on the wireless device. It carries out a secure authentication and key agreement mechanism with the trusted server at system startup. This allows the wireless device and the trusted server to validate each other's identities and to share the Source Key (SK_0). SK_0 is the source key that will be used by the Security Engine to open the secure log file and to derive the encryption keys for securing the confidentiality of the log records. A description of how a log file can be opened and closed securely using a shared key with a trusted server is described in [11]. The security services supported by the Security Engine are controlled and configured based on the information present in the policy configuration which provides the primary information source that the Security Engine consults for taking security decisions. The operation of the Security Engine in generating secure log records is described in Figure 3. The functions and notations used in Figure 3 are described below:

- *Log Record*$_i$: This is the ith log record generated by the Log Generator. Still this log record has to go to the Security Engine to be secured and stored in a secure log file.
- *Secure Log Record*$_i$: This is the secure log record produced by the Security Engine after applying the policy confidentiality and integrity rules on the ith log record (*Log Record*$_i$).
- $F_n(i)$: This is the nth field in the ith log record.

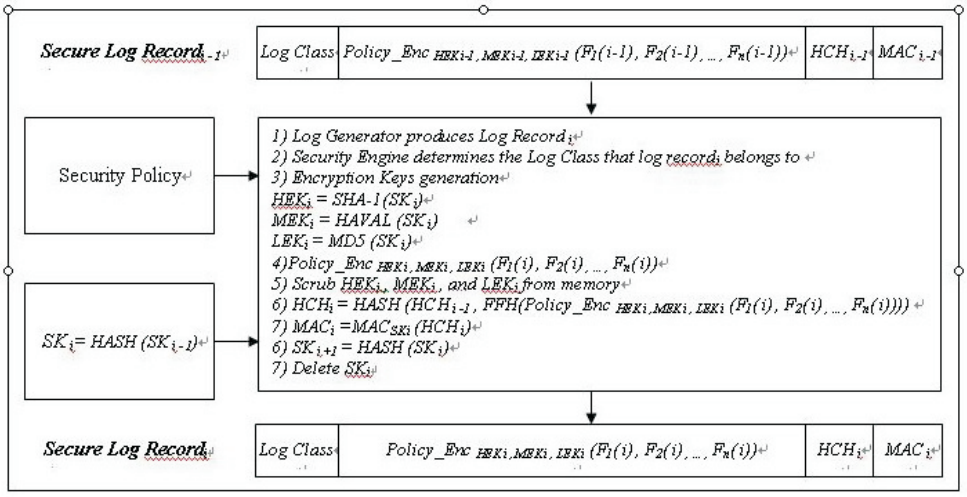

Figure 3: Secure log generation by the security engine component

- SK_i: This is the ith source key. SK_i is a hash of SK_{i-1}. It completely overwrites SK_{i-1} and derives the ith encryption keys used to secure the confidentiality of *Log Record*$_i$. Since SK_i is produced using a one-way hash function, an attacker capturing SK_i can't derive SK_{i-1}, SK_{i-2}, K SK_0.

- HEK_i, MEK_i, *and* LEK_i: These are the High, Medium, and Low encryption keys for encrypting the ith log record. They respectively represent the high, medium, and low security levels specified in PATRIOT's security policy. HEK_i is a 256-bit key. It is generated by hashing SK_i using a 256-bit-output hash function, such as SHA-1 [8]. MEK_i is a 192-bit key. This key is generated by hashing SK_i using a 192-bit-output hash function, such as HAVAL [13]. LEK_i is a 128-bit key. It is generated by hashing SK_i with a 128-bit-output hash function, such as MD5 [9].

- $Policy\_Enc_{HEK_i,MEK_i,LEK_i}(F_1(i), F_2(i), K, F_n(i))$: This function encrypts Log Recordi fields according to the rules specified in the security policy. The *High_Security*, *Medium_Security*, and *Low_Security* levels are provided using the HEK_i, MEK_i, and LEK_i encryption keys respectively. Note that some fields and field portions may not be encrypted. Moreover, different field portions may be encrypted using different encryption strengths. (See Figure 2). manufacturer. In this case the Security

- $FFH(Policy\_Enc_{HEK_i,MEK_i,LEK_i}(F_1(i), F_2(i), K, F_n(i)))$: This function extracts from the *Policy_Enc* function output, the secure fields whose Integrity Enforcement parameter is set to Yes in the security policy. These secure field values are concatenated and included in the hash chain that guarantees the integrity of the secure log file.

- HCH_i: This is the hash chain constructed by hashing the output of the FFH function and the hash chain entry of the previous secure log record (HCH_{i-1} of *Secure Log Record*$_{i-1}$). Since HCH_i includes $_{HCHi-1}$, it is possible to verify the integrity of all previous secure log records by only authenticating HCH_i. Initially, HCH_{-1} must be given a default value to start the hash chain.

- MAC_i: This is the MAC of HCH_i under the key SK_{i+1}. MAC_i is used to authenticate HCH_i.

The design of the Security Engine makes it impossible for an attacker to read or modify log entries generated before the time of the attack. If the attacker captures the secure log file after *Secure Log Record*$_i$ is generated, he will only be able to retrieve SK_{i+1} from the memory of the device. Since SK_{i+1} is derived using a one-way hash function from SK_i, the attacker will not be able to deduce SK_i and as a result the encryption keys HEK_i, MEK_i, and LEK_i derived from SK_i are also protected. By protecting SK_i, the privacy and confidentiality of the log records is assured and the authenticity of the hash chain is guaranteed.

5 Verification of the Secure Log File

PATRIOT allows the verification of the secure log file on the wireless device side. This is done by deploying a trusted verification software component on the wireless device at the verification time. The verification authority controlling and validating the operations of the wireless device must audit the deployment and verification operations. The verification authority supplies the verification component with the initial source key SK_0. With SK_0, the verification component can generate all subsequent

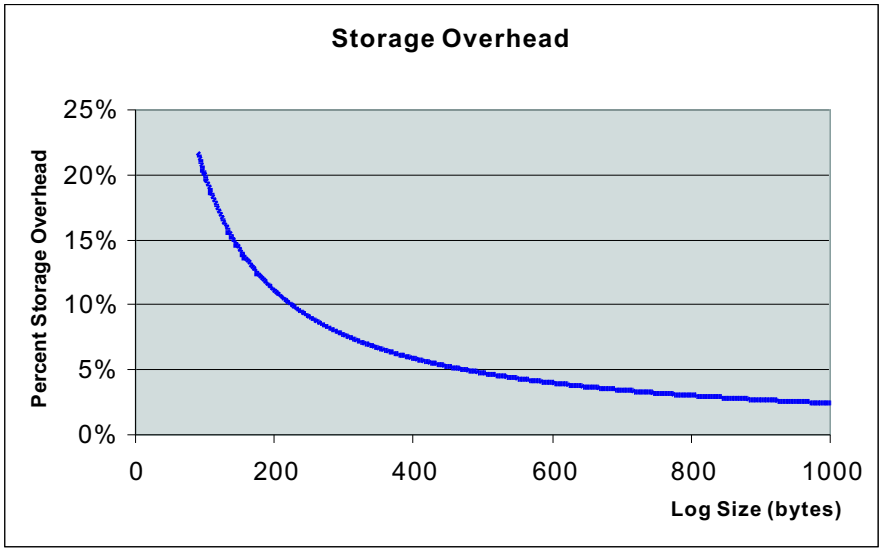

Figure 4: Percent storage overhead vs. the log record size

source and encryption keys. It checks the validity and authenticity of the hash chain using the *HCH* and *MAC* entries and makes sure that the log file doesn't contain any traces of a possible malicious attack. It should be noted here that the verification of the authenticity of the hash chain can be done without decrypting the log records. This is due to the fact that the hash chain depends in its construction on the encrypted fields of the log record.

6 Performance Analysis

This section presents a formal mathematical analysis of PATRIOT's performance. Most of the equations presented in this performance analysis are conducted in a platform-neutral manner without relying on any device hardware or operating system (unless specified otherwise in the text). In the rest of this section we will use the following notation:

1) F_j^T represents the size in bytes of the j^{th} field in the audit log.

2) $F_j^{E(k)}$ represents the number of encrypted bytes in the j^{th} field in the audit log based on the specifications of the k^{th} log class in the security policy.

3) $P(k)$ represents the probability that a generated log record complies with the criteria of the k^{th} log class in the security policy (satisfies the regular expression s pecified by the k^{th} log class in the security policy).

4) i_t represents the number of records generated in a time interval t.

5) $\mu_j(i_t)$ is the mean size of the contents of the j^{th} field in the i log records generated in the time interval t.

6) $\mu_j^{E(k)}$ is the average number of encrypted bytes of the j^{th} field in the i log records generated in time t and based on the specification of the k^{th} log class in the security policy.

7) $\mu_j^{EL(k)}$ is similar to $\mu_j^{E(k)}$ where the encryption mechanism is done using the low encryption level.

8) $\mu_j^{EM(k)}$ is similar to $\mu_j^{E(k)}$ where the encryption mechanism is done using the medium encryption level.

9) $\mu_j^{EH(k)}$ is similar to $\mu_j^{E(k)}$ where the encryption mechanism is done using the high encryption level.

10) *PDE* is the percent decrease in encryption operations resulting from the use of policy-based encryption mechanisms.

11) *PDE(t)* is the percent decrease in encryption operations at time t.

12) *PLE(t)* is the percentage of bytes encrypted according to the low encryption level in the time interval $[0-t]$.

13) *PME(t)* is the percentage of bytes encrypted according to the medium encryption level in the time interval $[0-t]$.

14) *PDH* represents the percent decrease in hashing operations resulting from the use of policy-based hashing mechanisms. *PHE(t)* is the percentage of bytes encrypted according to the high encryption level in the time interval $[0-t]$.

15) *PDH* represents the percent decrease in hashing operations resulting from the use of policy-based hashing mechanisms.

16) N represents the number of fields composing the log record.

17) C represents the number of log classes specified in the security policy.

6.1 Percent Decrease in Encrypted Bytes

The percent decrease in encrypted bytes, *PDE*, resulting from the use of content-based encryption mechanisms, compared to the traditional security approach which encrypts the whole contents of the log file, is given as follows:

$$PDE = \left(\begin{array}{l} \frac{(F_1^T - F_1^{E(1)} + F_2^T - F_2^{E(1)} + \cdots + F_N^T - F_N^{E(1)})}{(F_1^T + F_2^T + \cdots + F_N^T)} P(1) \\ + \\ \frac{(F_1^T - F_1^{E(2)} + F_2^T - F_2^{E(2)} + \cdots + F_N^T - F_N^{E(2)})}{(F_1^T + F_2^T + \cdots + F_N^T)} P(2) \\ + \cdots + \\ \frac{(F_1^T - F_1^{E(C)} + F_2^T - F_2^{E(C)} + \cdots + F_N^T - F_N^{E(C)})}{(F_1^T + F_2^T + \cdots + F_N^T)} P(C) \end{array} \right) \times 100$$

$$\Rightarrow PDE = \left(\frac{\sum_{k=1}^{C} (\sum_{j=1}^{N} (F_j^T - F_j^{E(k)}) P(k)}{\sum_{j=1}^{N} F_j^T} \right) \times 100$$

If the size of the fields composing the log record is not fixed, then F_j^T may have a different size in every generated log record and as a result $F_j^{E(k)}$ may also vary. So to find PDE, we have to take into consideration the number of records i_t generated in a given time interval t. Let $F_j^T(i)$ be the size of the j^{th} field in i^{th} generated log record and $F_j^{E(k)}(i)$ be the number of encrypted bytes in the j^{th} field of the i^{th} generated log record. The encryption is done according to the specification of the k^{th} log class in the security policy. We have that

$$\mu_j(i_t) = \frac{(F_j^T(1) + F_j^T(2) \cdots + F_j^T(i_t))}{i_t}$$

and

$$\mu_j^{E(k)}(i_t) = \frac{F_j^{E(k)}(1) + F_j^{E(k)}(2) + \cdots + F_j^{E(k)}(i_t))}{i_t}$$

$$\Rightarrow PDE(t) = \left(\begin{array}{l} \frac{M_1}{(\mu_1(i_t) + \mu_2(i_t) + \cdots + \mu_N(i_t))} P(1) + \\ \frac{M_2}{(\mu_1(i_t) + \mu_2(i_t) + \cdots + \mu_N(i_t))} P(2) + \cdots + \\ \frac{M_3}{(\mu_1(i_t) + \mu_2(i_t) + \cdots + \mu_N(i_t))} P(C) \end{array} \right) \times 100$$

$$\Rightarrow PDE(t) = \left(\frac{\sum_{k=1}^{C} (\sum_{j=1}^{N} (\mu_j(i_t) - \mu_j^{E(k)}(i_t)))}{\sum_{j=1}^{N} \mu_j(i_t)} \right) \times 100$$

$$M_1 = \mu_1(i_t) - \mu_1^{E(l)}(i_t) + \mu_2(i_t) - \mu_2^{E(l)} + \cdots + \mu_N(i_t) - \mu_N^{E(l)}(i_t)$$

$$M_2 = \mu_1(i_t) - \mu_1^{E(2)}(i_t) + \mu_2(i_t) - \mu_2^{E(2)}(i_t) + \cdots + \mu_N(i_t) - \mu_N^{E(2)}(i_t)$$

$$M_3 = \mu_1(i_t) - \mu_1^{E(C)}(i_t) + \mu_2(i_t) - \mu_2^{E(C)}(i_t) + \cdots + \mu_N(i_t) - \mu_N^{E(C)}(i_t)$$

By replacing the values of $\mu_j(i_t)$ and $\mu_j(E(k)(i_t)$in the above equation, $PDE(t)$ can be expressed as follows:

$$\Rightarrow PDE(t) = \left(\frac{\sum_{k=1}^{C} (\sum_{j=1}^{N} (\sum_{s=1}^{i_t} (F_j^T(s) - F_j^{E(k)}(s))))}{\sum_{j=1}^{N} \sum_{s=1}^{i_t} F_j^T(s)} P(k) \right)$$

Example 1 *Let* $N = 6, C = 4, i_t = 10, P(1) = 0.2, P(2) = 0.4, P(3) = 0.3, P(4) = 0.1$ *Consider the matrices* $F, CP, FPC1, FPC2, FPC3$ *and* $FPC4$ *where:*

- $F[i,j] = F_j^T(i)$
- $CP[i,j]$ *is the percent encryption in the* j^{th} *field based on the specifications of the* i^{th} *log class.*
- $FPC1[i,j] = F_j^T(i) - F_j^{E(1)}(i)$
- $FPC2[i,j] = F_j^T(i) - F_j^{E(2)}(i)$
- $FPC3[i,j] = F_j^T(i) - F_j^{E(3)}(i)$
- $FPC4[i,j] = F_j^T(i) - F_j^{E(4)}(i)$

$$F = \left| \begin{array}{cccccc} 35 & 70 & 15 & 55 & 261 & 22 \\ 72 & 44 & 13 & 80 & 365 & 20 \\ 33 & 26 & 70 & 55 & 177 & 16 \\ 45 & 76 & 32 & 23 & 209 & 14 \\ 33 & 99 & 83 & 30 & 198 & 20 \\ 24 & 60 & 44 & 35 & 277 & 45 \\ 77 & 66 & 32 & 70 & 160 & 31 \\ 90 & 12 & 50 & 34 & 255 & 12 \\ 43 & 66 & 11 & 54 & 302 & 33 \\ 61 & 44 & 71 & 66 & 422 & 77 \end{array} \right| \quad CP = \left| \begin{array}{cccccc} 25 & 40 & 22 & 44 & 22 & 89 \\ 75 & 38 & 50 & 50 & 41 & 72 \\ 23 & 77 & 16 & 50 & 10 & 73 \\ 44 & 52 & 22 & 42 & 12 & 57 \end{array} \right|$$

$$\Rightarrow \sum_{j=1}^{N} \sum_{s=1}^{i_t} F_j^T(s) = 4915$$

$$\sum_{j=1}^{N} (\sum_{s=1}^{i_t} (F_j^T(s) - F_j^{E(1)}(s))) = 3412.23$$

$$\sum_{j=1}^{N} (\sum_{s=1}^{i_t} (F_j^T(s) - F_j^{E(2)}(s))) = 2569.35$$

$$\sum_{j=1}^{N} (\sum_{s=1}^{i_t} (F_j^T(s) - F_j^{E(3)}(s))) = 3570.84$$

$$\sum_{j=1}^{N} (\sum_{s=1}^{i_t} (F_j^T(s) - F_j^{E(4)}(s))) = 3612.64$$

$$\sum_{k=1}^{C} (\sum_{j=1}^{N} (\sum_{s=1}^{i_t} (F_j^T(s) - F_j^{E(k)}(s)))) P(k) = 3142.702$$

$$\Rightarrow PDE = 63.94\%$$

6.2 Percent Decrease in Hashing Operations

In this section we will calculate the percent decrease in hashing operations resulting from the use of policy-based hashing mechanisms.

Let $R^{H(k)}$ represents the number of fields entering the hash chain by having their Integrity Enforcement parameter set to Ÿesäs specified by the k^{th} log class in the security policy (see Figure 2).

$$PDH\_record = \left(\frac{N_1}{N}\right) \times 100$$
$$\Rightarrow PDH\_record = \left(\frac{\sum_{k=1}^{C}(N - R^{H(k)}) \times P(k)}{N}\right) \times 100$$
$$N_1 = ((N - R^{H(1)}) \times P(1) + \cdots + (N - R^{H(C)}) \times P(C))$$

Example 2 *Let* $C = 4, N = 6, R = [4\ 3\ 1\ 2]$ *where* $R[i] = R^{H(i)}$
$P = [0.2\ 0.4\ 0.3\ 0.1]$ *where*
$P[i] = P(i) \Rightarrow PDH\_record = 58.33\%$

6.3 Performance Gain

In Section 6.1 we calculated the percent decrease in encrypted bytes resulting from the use of content-based encryption mechanisms. In this section we will calculate the performance gain achieved due to applying multi-level, fine-grained encryption.

Let:

1) $F_j^{EL(k)}(i)$ be the number of encrypted bytes in the j^{th} field of the i^{th} generated log record. The encryption mechanism is done using the low encryption level and based on the specification of the k^{th} log class in the security policy.

2) $F_j^{EM(k)}(i)$ is similar to $F_j^{EL(k)}(i)$ where the encryption mechanism is done using the medium encryption level.

3) $F_j^{EH(k)}(i)$ is similar to $F_j^{EL(k)}(i)$ where the encryption mechanism is done using the high encryption level.

$$\mu_j^{EL(k)}(i_t) = \frac{(F_j^{EL(k)}(1) + F_j^{EL(k)}(2) + \cdots + F_j^{EL(k)}(i_t))}{i_t}$$
$$\mu_j^{EM(k)}(i_t) = \frac{(F_j^{EM(k)}(1) + F_j^{EM(k)}(2) + \cdots + F_j^{EM(k)}(i_t))}{i_t}$$
$$\mu_j^{EH(k)}(i_t) = \frac{(F_j^{EH(k)}(1) + F_j^{EH(k)}(2) + \cdots + F_j^{EH(k)}(i_t))}{i_t}$$
$$\Rightarrow PLE(t) =$$
$$\left(\begin{array}{l} \frac{(\mu_1^{EL(1)}(i_t)+\mu_2^{EL(1)}(i_t)+\cdots+\mu_N^{EL(1)}(i_t))}{(\mu_1(i_t)+\mu_2(i_t)+\cdots+\mu_N(i_t))}P(1) \\ + \\ \frac{(\mu_1^{EL(2)}(i_t)+\mu_2^{EL(2)}(i_t)+\cdots+\mu_N^{EL(2)}(i_t))}{(\mu_1(i_t)+\mu_2(i_t)+\cdots+\mu_N(i_t))}P(2) \\ +\cdots+ \\ \frac{(\mu_1^{EL(C)}(i_t)+\mu_2^{EL(C)}(i_t)+\cdots+\mu_N^{EL(C)}(i_t))}{(\mu_1(i_t)+\mu_2(i_t)+\cdots+\mu_N(i_t))}P(C) \end{array}\right) \times 100$$
$$\Rightarrow PLE(t) = \left(\frac{\sum_{k=1}^{C}\sum_{j=1}^{N}\mu_j^{EL(k)}(i_t)}{\sum_{j=1}^{N}\mu_j(i_t)}\right) \times 100$$

Simularly

$$\Rightarrow PME(t) = \left(\frac{\sum_{k=1}^{C}\sum_{j=1}^{N}\mu_j^{EM(k)}(i_t)}{\sum_{j=1}^{N}\mu_j(i_t)}\right) \times 100$$
$$\Rightarrow PHE(t) = \left(\frac{\sum_{k=1}^{C}\sum_{j=1}^{N}\mu_j^{EH(k)}(i_t)}{\sum_{j=1}^{N}\mu_j(i_t)}\right) \times 100$$

Let X, Y, and Z be the cost of an encryption operation using the low, medium, and high encryption levels respectively. This cost may be the number of CPU cycles to perform an encryption operation or the RAM footprint consumed by an encryption operation, or a function of both. It should be noted that the values of X, Y, and Z are platform-dependent where $0 < X < Y < Z$ since the complexity of encryption is proportional to the size of the encryption key. Assume that the traditional security approach of securing audit logs uses an encryption strength equivalent to the medium encryption level used in PATRIOT.

The performance gain G at time t, resulting from the use of content-based encryption mechanisms, relative to the traditional approach is given as follows:

$$\begin{aligned} G(t) &= \left(\frac{Y_1}{Y}\right) \times 100 \\ Y_1 &= Y - ((PLE(i_t)/100 \times X + (PME(i_t)/100) \times Y \\ &\quad + (PHE(i_t)/100) \times Z \end{aligned}$$

Example 3 *Let* $N = 6, C = 4, i_t = 10, P(1) = 0.2,$ $P(2) = 0.4, P(3) = 0.3, P(4) = 0.1$. *Consider the matrices* $F, CL, CM, CH, FELC1, FELC2, FELC3, FELC4,$ $FEMC1, FEMC2, FEMC3, FEMC4, FEHC1,$ $FEHC2, FEHC3$, *and* $FEHC4$ *where:*

- $F[i,j] = F_j^T(i)$ *(same as in Example 1)*
- $CL[i,j]$ *is the percent of low encryption operations in the* j^{th} *field based on the specifications of the* i^{th} *log class.*
- $CM[i,j]$ *is the percent of medium encryption operations.*
- $CH[i,j]$ *is the percent of high encryption operations.*
- $FELC1[i,j] = F_j^{EL(1)}(i)$ $FELC2[i,j] = F_j^{EL(2)}(i)$
- $FELC3[i,j] = F_j^{EL(3)}(i)$ $FELC4[i,j] = F_j^{EL(4)}(i)$
- $FEMC1[i,j] = F_j^{EM(1)}(i)$ $FEMC2[i,j] = F_j^{EM(2)}(i)$
- $FEMC3[i,j] = F_j^{EM(3}(i)$ $FEMC4[i,j] = F_j^{EM(4)}(i)$
- $FEHC1[i,j] = F_j^{EL(1}(i)$ $FEHC2[i,j] = F_j^{EL(2)}(i)$
- $FEHC3[i,j] = F_j^{EL(3)}(i)$ $FEHC4[i,j] = F_j^{EL(4)}(i)$

$$CL = \left| \begin{array}{c} 5\ 12\ 10\ 0\ 22\ 30 \\ 30\ 30\ 5\ 0\ 20\ 5 \\ 8\ 20\ 6\ 5\ 8\ 33 \\ 5\ 10\ 2\ 2\ 6\ 20 \end{array} \right| CM = \left| \begin{array}{c} 10\ 13\ 2\ 22\ 0\ 0 \\ 10\ 4\ 30\ 20\ 1\ 30 \\ 5\ 15\ 5\ 20\ 0\ 20 \\ 4\ 25\ 10\ 15\ 5\ 7 \end{array} \right|$$

$$CH = \left| \begin{array}{c} 10\ 15\ 10\ 22\ 0\ 59 \\ 35\ 4\ 15\ 30\ 20\ 37 \\ 10\ 42\ 5\ 25\ 2\ 20 \\ 35\ 17\ 10\ 25\ 1\ 30 \end{array} \right|$$

$$\Rightarrow \quad PLE = 14.20\% \qquad PME = 7.00\%$$
$$PHE = 14.85\% \qquad G = \mathbf{65.55}\%$$

6.4 Storage Overhead

The storage overhead per log record in PATRIOT depends on the size of the log class identifier in the security policy and the size of the hash chain and MAC entries in the log record. Consider the following notations:

1) *SizeOf*($LGCI$) represents the size of the log class identifier in bytes.

2) *SizeOf*(HCH) represents the size of the hash chain entry in bytes.

3) *SizeOf*(MAC) represents the size of the MAC entry in bytes.

The percent storage overhead (PSO) per log record is given as follows:

$$PSO/_{record} = \left(\frac{S_1}{S_1 + \sum_{j=1}^{N} F_j^T} \right) \times 100$$
$$S_1 = SizeOf(LGCI) + SizeOf(HCH) + SizeOf(MAC)$$

The size of the log class identifier is implementation-dependent since it depends on the number of log classes defined in the security policy. In general, $SizeOf(LGCI)$ is given as follows:

$$SizeOf(LGCI) = \frac{\log_2(C)}{8}$$

$SizeOf(HCH)$ and $SizeOf(MAC)$ depend on the hash and MAC algorithms respectively.

Example 4 *Let* $SizeOf(LGCI) = 1$ *byte,* $SizeOf(HCH) = 16$ *bytes, and* $SizeOf(MAC) = 8$ *bytes. Assuming that the minimum value of* $\sum_{j=1}^{N} F_j^T$ *is 90 bytes for a log record consisting of 6 log fields* ($N = 6$) *the maximum storage overhead is 21.73 %. The percent storage overhead per log record versus the log record size is shown in Figure 4.*

7 PATRIOT Implementation

PATRIOT was designed in a platform-neutral manner and its design features are optimized for implementation on a wide range of available wireless client platforms. A sample simulation implementation for PATRIOT was developed for a Pocket PC client device using the .NET Compact framework 2.0 specifications.

In this Section, a brief description of the implementation along with the technologies, tools and devices used are presented. We begin with an overview of the .Net Compact Framework 2.0 followed by a description of the emulator and the Pocket PC device used. In addition, the implementation software components and classes are detailed along with their functionalities. Finally a description of the performance of the parsing, encryption, decryption, and hashing operations is provided.

7.1 .NET Compact Framework Overview

The .Net Compact Framework is Microsoft's platform for developing mobile applications on low profile devices. It targets personal digital assistants (PDAs), mobile phones and set-top boxes. PDAs are usually referred to as Pocket PCs. The .NET Compact Framework features a low memory footprint of 0.5 MB of RAM and 1.35 MB ROM on

Windows Mobile for Pocket PC 2003 or Windows CE .NET Devices with typical application sizes of 5-100 KB.

7.2 Emulation and Testing Environment

PATRIOT's simulation was developed using Microsoft Visual Studio 2005 beta 2 [7] integrated development environment. Visual Studio ships with embedded Pocket PC emulators that can be easily utilized to test and debug mobile .NET applications. It is even possible to configure the environment variables and the specifications of the virtual device such as the total memory, memory allocated to programs, serial ports and network addresses. In the experiments that have been conducted in this work, all the specifications were left as default except for the total memory which was set to 64 MB.

The simulation application was later tested on an HP iPAQ rx3115 Pocket PC device with a Samsung S3C 2440 processor (300MHz) and 56 MB of RAM. Windows Mobile 2003 is the operating system used. The device comes with integrated WiFi capabilities which were useful to test the interactions with a server over the wireless network.

7.3 Software Components

The code was totally written in the C# programming language. The application consists of two projects put together in one solution. One is the library that has all the functionality organized within .Net namespaces and classes. The root namespace is "Library" and all the functionality needed is organized in appropriate namespaces there under. The other part is the user-friendly interface which provides the tester with an easy way to interact with and test the system. The system is an emulation of a secure logging system. To emulate a logger, a text file that contains a large number of log records is used. The

Table 1: Performance results

	Encryption Time	*Encryption Rate*	*Encryption Time Per record*	*Decryption Time*	*Decryption Rate*	*Encryption Time Per record*
Policy-based logger	22 sec	331.7 kbps	23.45 ms	38 sec	192.0 kbps	40.51 ms
Traditional logger (record basis)	32 sec	228.0 kbps	34.12 ms	58 sec	125.8 kbps	61.83 ms
Traditional logger (field basis)	42 sec	173.7 kbps	44.77 ms	80 sec	91.2 kbps	85.28 ms

system reads the text file one record at a time and feeds the *PolicyEncryptor* component. All this is done after loading and parsing the security policy. The decryption process follows almost the same steps by reading the secured log file line by line, each line corresponding to one record, and then feeding the *PolicyDecryptor* component which is responsible of reconstructing the original data according to the specifications provided by the security policy.

7.4 Procedure Overview

The first step in order to launch the secure log emulator is to read the policy from the corresponding text file and send it to the *PolicyParser* component which outputs the representing Policy object. This policy, along with the initial keys, is used to initialize the *Policy Encryptor* component. Then, the Encryptor will be ready to receive requests to encrypt / decrypt records from the logger emulator. Each record is being read and later sent to the Encryptor which in turn checks for the log class of the current record after referring to the policy already loaded. As before, the record log class matching is done using regular expressions.

7.5 Analysis and Performance Comparison

A log file of around 900 records each consisting of three fields is used. The total size of the log file is 890 KB. These fields are: date, time and description of the event that is to be logged. In the tests, the classes are being distinguished based on date or time patterns. For example, one class might be for events that happens in a certain month of the year because of the sensitivity of operations that are undertaken yearly in that specific month, such as the yearly budget revision for an organization. The comparison is made with a traditional security system that secures the log file by encrypting all its contents. In this method the whole record is encrypted and hashed.

One thing to consider is the parsing time of the policy, which does not exist in the traditional logger. This process, the reading of the file and parsing the policy, took in the environment discussed so far around 370 milliseconds total. This is done one time at the starting-up of the system. Clearly, it is not a real deficiency for the policy driven system since it happens once at startup. For the traditional security method, two cases are studied: securing the log on record basis or on field basis. Table 1 presents some performance measurements conducted on both the traditional security mechanism and the policy-based security system for around 900 records. The traditional method was undertaken in Medium level security (192-bit key) while the policy based approach is adaptive. For each class of records, each field or even part of a field can be secured with the four previously discussed security levels. The hashing rate was found to be 1085.9 kbps.

From Table 1 it is possible to notice clearly the improvement that is achieved by just encrypting/ decrypting what is really needed to be secured. Policy-based numbers differ slightly depending on the policy being used. In general, the results show an improvement of 48% over the field basis method and 31% over the record basis encryption.

8 Conclusion

In this paper we presented PATRIOT, a policy-driven security architecture for protecting the confidentiality and integrity of logging systems on wireless devices. PATRIOT employs a flexible, fine-grained, and multi-level encryption methodology that suits the diverse nature and capabilities of existing wireless devices. The architecture is based on well-known cryptographic protocols and is designed in a platform-neutral manner that makes it deployable on a wide range of wireless devices and operating systems. The paper presented a description of PATRIOT's design and architecture and presented using a formal mathematical performance analysis and a simulation on an HP iPAQ rx3115 Pocket PC device the advantages of a policy-based security solution for securing audit logs on wireless devices.

References

[1] M. Bellare and B. Yee, *Forward Integrity for Secure Audit Logs*, Technical Report, Computer Science and

Engineering Department, University of California at San Diego, Nov. 1997.
[2] J. Daemen and V. Rijmen, "Rijndael, the advanced encryption standard," *Dr. Dobb's Journal*, vol. 26, no. 3, pp. 137-139, Mar. 2001.
[3] W. Itani and A. Kayssi, "J2ME end-to-end security for M-Commerce," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, pp. 2015-2020, 2003.
[4] W. Itani and A. Kayssi, "SPECSA: a scalable, policy-driven, extensible, and customizable security architecture for wireless enterprise applications," in *Proceedings of the IEEE IPCCC Workshop on Information Assurance (WIA04)*, pp. 753-759, Phoenix, Arizona, Apr. 2004.
[5] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
[6] *Microsoft's .Net Compact Framework Home Page*, http://msdn.microsoft.com/smartclient/netcf/default.asp
[7] *Microsoft's Visual Studio 2005 Home Page*, http://msdn.microsoft.com/vs2005 /default.aspx.
[8] National Institute of Standards and Technology, *Secure Hash Standard*, Federal Information Processing Standards, Publication 180-1, 1995.
[9] R. Rivest, *The MD5 Message-digest Algorithm*, RFC 1321, 1992.
[10] B. Schneier and J. Kelsey, "Remote auditing of software outputs Using a Trusted Coprocessor," *Future Generation Computer Systems13*, vol. 1, pp. 9-18, 1997.
[11] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics", *ACM Transactions on Information and System Security*, vol. 2, no. 2, pp. 159-196, May 1999.
[12] R. Snodgrass, S. Yao, and C. Collberg, "Tamper detection in audit logs", in *Proceedings of the 30th VLDB Conference, Toronto*, pp. 504-515, Canada, 2004.
[13] Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL–A one-way hashing algorithm with variable length of output," in *Auscrypto '92*, LNCS 718, pp. 83-104, 1992.

Wassim Itani was born in Beirut, Lebanon in 1978. He holds a Master's degree in Computer and Communications Engineering from the American University of Beirut (AUB). Since his graduation from AUB in June 2003, he worked as a research assistant in the Department of Electrical and Computer Engineering. Wassims current research interests include policy-based networking, mobile computing and cryptographic protocols performance.

Ayman Kayssi was born in Lebanon in 1967. He received his BE with distinction in 1987 from the American University of Beirut, Lebanon and the MSE in 1989 and PhD in 1993 from the University of Michigan, Ann Arbor, USA, all in electrical engineering. He is currently professor and chairman of electrical and computer engineering at the American University of Beirut, where he has been working since 1993. His research and teaching interests are in the areas of internet engineering, wireless networking, CAD for VLSI, modeling and simulation.

Ali Chehab received his Bachelor degree in EE from the American University of Beirut (AUB) in 1987, the Master's degree in EE from Syracuse University, and the PhD degree in ECE from the University of North Carolina at Charlotte, in 2002. From 1989 to 1998, he was a lecturer in the ECE Department at AUB. He rejoined the ECE Department at AUB as an assistant professor in 2002. His research interests are VLSI design and test, mobile agents, and wireless security.

Camille Gaspard received his Bachelor degree in Computer Engineering from Aleppo University - Syria in 2004. He is now preparing his Masters degree in Computer and Communications Engineering at the American University of Beirut (AUB). His research interests are network security, cryptography, semantic web and mobile agents.